On Privacy-Preserving Cloud Auction

Conference Paper · September 2016

DOI: 10.1109/SRDS.2016.045

Zhili Chen
University of Science and Technology of China
42 PUBLICATIONS 187 CITATIONS
SEE PROFILE

Lin Chen
Université Paris-Sud 11
116 PUBLICATIONS 771 CITATIONS
SEE PROFILE

2016 IEEE 35th Symposium on Reliable Distributed Systems

On Privacy-preserving Cloud Auction

Zhili Chen[1], Lin Chen[2], Liusheng Huang[3,4], Hong Zhong[1]

[1]School of Computer Science and Technology, Anhui University, Hefei, China
[2]Lab. Recherche Informatique (LRI-CNRS UMR 8623), Univ. Paris-Sud, 91405 Orsay, France
[3]School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
[4]Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, China
Email: zlchen3@ustc.edu.cn, chen@lri.fr, lshuang@ustc.edu.cn, zhongh@mail.ustc.edu.cn

***Abstract*—Due to perceived fairness and allocation efficiency, cloud auctions for resource allocation and pricing have recently attracted significant attention. As an important economic property, truthfulness makes bidders reveal their true valuations for cloud resources to maximize their utilities. However, disclosure of one's true value causes numerous security vulnerabilities. Therefore, privacy-preserving cloud auctions are called for to prevent such information leakage. In this paper, we demonstrate how to perform privacy-preserving auctions in clouds that do not leak any information other than the auction results to anyone. Specifically, we design a privacy-preserving cloud auction framework that addresses the challenges posed by the cloud auction context by leveraging the techniques in garbled circuits and homomorphic encryption. As foundations of our privacy-preserving cloud auction framework, we develop *data-oblivious* cloud auction algorithm and basic operations (e.g., comparison, swapping etc.), such that the execution path does not depend on the input. In practical systems with a large number of users and constrained resources, we develop an improved version with a computational complexity of $O(n \log^2 n)$ in the number of bidders n. We further fully implement our framework and theoretically and experimentally show that it preserves privacy by incurring only limited computation and communication overhead.**

I. Introduction

Due to perceived fairness and allocation efficiency, auctions are among the best-known market-based allocation mechanisms and have recently attracted significant research attention on cloud resource allocation and pricing from both academia and industry. For example, Amazon EC2 has integrated an auction mechanism called *Spot Instance* to allocate the virtual machines (VMs), termed as instances. A large body of work has then developed enhanced auction mechanisms satisfying various economic properties such as truthfulness, profit maximization, and social efficiency (cf. [1], [2] and references therein). However, none of these mechanisms has considered *privacy preservation* in their cloud auction design, thus leaving bid information largely exposed to auction participants (i.e., auctioneer and bidders) as well as other users. The disclosure of bid information in a cloud auction may lead to severe information leakage that can be exploited by different entities.

- For the auctioneer, it can simply adapt its pricing strategy based on the bidders' bids to obtain extra profit.
- For the bidders, since VM instances are usually granted for a limited time block, cloud auctions are executed periodically. Through learning the historical bids of others (possibly leaked from the auctioneer), bidder may get to know others' willingness to pay, and choose to bid untruthfully to get extra profit, thus tampering with the truthfulness of the whole auction.
- For an attacker not participating the auction, once obtaining bid information, it can impair the auction process by submitting a bid that cannot win the auction, but will increase the price paid by the winners.

Consequently, the information leak by an auction without privacy preservation not only reveals the private information of bidders, but also tampers with economic properties of the auction such as truthfulness and social efficiency.

Motivated by the above observation, in this paper we develop a privacy-preserving cloud auction framework that runs without disclosing anything about the bids except what is revealed from the auction results. We focus on a truthful cloud auction mechanism proposed recently [3], where a cloud provider provides various computing resources to a large number of heterogeneous users. Architecturally, our privacy-preserving cloud auction framework is depicted in Fig. 1: Cloud users (bidders) want to keep their bids private during the auction. They send encrypted bids to a cloud provider (CloudPdr) who runs the privacy-preserving auction, with the intervention of a *crypto-service provider* (CryptPdr), whose role is to enable private-preserving computations in the auction. As long as CloudPdr and CryptPdr do not collude, our solution ensures the following security properties:

- No bidder can learn anything about the bids of other bidders, except the auction results of the current run (i.e., the IDs, clearing prices, and requested bundles of VM instances, of the winners);
- Neither CloudPdr nor CryptPdr can learn anything about the bids of all bidders, except the auction results of the current run;
- Any user not participating the auction cannot learn anything about the bids of all bidders, except the auction results of the current run.

Compared with other conventional auctions, privacy preserving design in cloud auctions has the following specific challenges. The first one is the heterogeneity in both computing resources (VM instances) and user preferences. It is not clear how privacy-preserving auction can be performed under such heterogeneous environment. Secondly, cloud users may

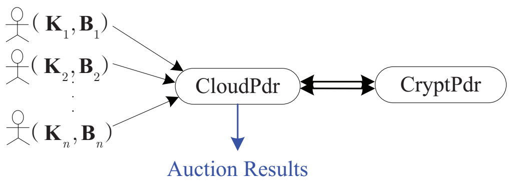

$(\mathbf{K}_1,\mathbf{B}_1)$
$(\mathbf{K}_2,\mathbf{B}_2)$
$(\mathbf{K}_n,\mathbf{B}_n)$
CloudPdr
CryptPdr
Auction Results

Fig. 1. Privacy-preserving cloud auction architecture: The CloudPdr computes the auction results with the help of CryptPdr, without disclosing anything about the bids $\{(\mathbf{K}_i, \mathbf{B}_i)\}_{i=1}^n$ except the auction results to anyone.

not stay online during the entire auction duration. In other words, a user may switch offline after having submitted its bid. Last but not least, privacy-preserving auctions should be efficient, and scale nicely in both computation and communication so as to support auctions with a large number of users.

The above challenges require the auction algorithm be *data-oblivious*, meaning that its execution path should not depend on the input, i.e. user bids, and all bids remain encrypted throughout the auction, and furthermore the secure auction be efficient enough. We thus design our secure cloud auction protocol by leveraging tools in garbled circuits and homomorphic encryption, and design the data-oblivious auction algorithm and basic operations, which are easily converted into a circuit used in the garbled circuit protocol. The complexity of our privacy-preserving cloud auction is $O(n \log^2 n)$ which is within a logarithmic factor of the complexity of the original auction algorithm $O(n \log n)$.

Our contributions are articulated as follows.

- By leveraging the techniques in garbled circuits and homomorphic encryption, we propose a privacy-preserving cloud auction framework for the truthful cloud auctions, and then design the data-oblivious cloud auction algorithm (based on the proposition in [3]) and the needed data-oblivious basic operations such as comparison, swap, and arithmetical operations. These algorithms are implemented as an auction circuit, and then used in our cloud auction framework, yielding a privacy-preserving cloud auction protocol with a complexity $O(n^2)$ in the number of bidders n. Finally, we formally prove privacy preservation of our protocol in the sense of cryptography.
- We further improve our protocol, such that its complexity becomes $O(n \log^2 n)$ when n is large and the computing resource is constraint, while achieves provable privacy as our original protocol.
- We implement our solution, and conduct extensive simulation experiments. Our evaluation shows that our solution incurs limited computation and communication overhead.

To our knowledge, our proposition is the first privacy preserving solution for cloud auctions.

The rest of the paper is organized as follows. Section II gives the problem formulation. Technical preliminaries are provided in Section III. We present our protocol, and formally prove its privacy preservation in Section IV. In Section V, we improve our protocol to reduce its complexity in systems with a large number of users. In Section VI, we implement both our original and improved protocols, and evaluate the performance in terms of computation and communication overhead. Section VII briefly reviews related work and situate our work in the literature. Finally, the paper is concluded in Section VIII.

TABLE I
NOTATIONS FOR CLOUD AUCTION

m, n	numbers of VM types and bidders
$(\mathbf{K}_j, \mathbf{B}_j)$	bid of bidder j
k_j^i	number of type-i VM instances requested by j
b_j^i	per-instance bid of type-i VMs of j
b_j	$b_j = \sum_{i=1}^m k_j^i b_j^i$, j's total bid for its bundle
w_i	type-i VM instances' weight
ω_j	j's total amount of weighted VM resource for its bundle
b_j^*	j's bid per unit of weighted VM resource for its bundle
p_j	j's price for its bundle
$\mathbf{K}_j$	$\mathbf{K}_j = \{k_j^i\}_{i=1}^m$, VM bundle requested by j
$\mathbf{B}_j$	$\mathbf{B}_j = \{b_j^i\}_{i=1}^m$, per-instance bid set of j
$\mathbf{K}$	$\mathbf{K} = \{\mathbf{K}_j\}_{j=1}^n$, global bundle set
$\mathbf{B}$	$\mathbf{B} = \{\mathbf{B}_j\}_{j=1}^n$, global per-instance bid set
$\mathbf{P}$	$\mathbf{P} = \{p_j\}_{j=1}^n$, global payment set

II. Problem Formulation

A. System Model

We consider a cloud auction setting with heterogeneous VMs, where a cloud provider (CloudPdr) offers m types of VM instances characterized by a number of VM attributes such as CPU and memory (e.g., small, medium, large), operating systems (e.g., Linux, Microsoft Windows), etc. There are k_i instances of type-i VMs. There are n bidders requesting VMs with each bidder j ($1 \le j \le n$) asking for k_j^i instances of type-i VMs. Let $\mathbf{K}_j = \{k_j^i\}_{i=1}^m$ denote the VM bundle requested by bidder j, it holds that $\mathbf{K}_j \in [0..k_1] \times [0..k_2] \times \cdots \times [0..k_m]$. Notations for the cloud auction are summarized in Tab. I. Especially, we set $b_j^i = 0$ if bidder j does not ask for type-i VMs, and set $p_j = 0$ if bidder j does not win the auction.

B. Cloud Auction Mechanism

For the above resource provisioning problem in clouds, a computationally efficient truthful auction mechanism has been proposed in [3], which we sketch its three major steps as follows.

- **Step 1: Bid Ranking.** Each bidder j computes the total bid b_j from $(\mathbf{K}_j, \mathbf{B}_j)$, and sets b_j^* as follows:

$$b_j^* = \frac{b_j}{\omega_j}, \tag{1}$$

where $\omega_j \triangleq \sqrt{\sum_{i=1}^m k_j^i w_i}$ with w_i being the weight of type-i VM instances assigned by the CloudPdr, and the instance weights w_i's are introduced to distinguish between different VM instances, e.g., a VM instance with higher CPU speed or larger memory has a greater weight and thus requests a higher bid.

CloudPdr sorts the bidders in descending order of b_j^*, where b_j^* can intuitively be regarded as the bid per unit of weighted VM resource by bidder j.

- **Step 2: VM Allocation.** According to the sorted bidder list, for bidder j, if its required VM resources are available, i.e.

$$\sum_{t=1}^{j-1} k_t^i x_t + k_j^i \leq k_i, \forall i \in \{1, 2, \cdots, m\}, \tag{2}$$

then its request is satisfied ($x_j = 1$); otherwise, $x_j = 0$.

- **Step 3: Pricing.** CloudPdr applies the following pricing rules:
  - A denied bidder pays 0;
  - A bidder receiving the desired bundle pays the critical value;
  - A bidder having no critical bidder pays 0.

  A critical bidder of bidder j, denoted as bidder c, is the first bidder following bidder j that has been denied but would become a winner without j. The critical value v_c^j of bidder j is the minimum value bidder j should bid in order to win the auction. It can be shown that $v_c^j = b_c^* \omega_j$. CloudPdr charges each winning bidder j ($x_j = 1$) the following amount:

$$p_j = v_c^j = b_c^* \omega_j. \tag{3}$$

C. *Privacy-preserving Cloud Auction Design*

In our work, we aim at designing privacy-preserving cloud auctions that do not leak any information to any entity other than the auction results. By auction results we mean the IDs, clearing prices, and VMs allocated to the winners, which should be published after the auction. By entities we refer to CloudPdr, CryptPdr, bidders and other users not participating the auction. Specifically, the following security requirements should be satisfied:

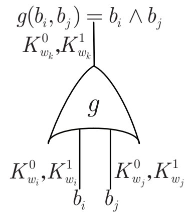
- No bidder can learn anything about the bids of other bidders, except the auction results of the current run;
- Neither CloudPdr nor CryptPdr can learn anything about the bids of all bidders, except the auction results of the current run;
- Any user not participating the auction cannot learn anything about the bids of all bidders, except the auction results of the current run.

In our work, we assume that CloudPdr and CryptPdr do not collude with each other. We focus on the *semi-honest* (or *honest but curious*) threat model where CloudPdr and CryptPdr follow the protocols we develop but may analyze protocol transcripts to infer additional information.

III. Technical Preliminaries

A. *Garbled Circuits*

Garbled circuits (a.k.a Yao's protocol) [4], [5] is a generic technique for secure multi-party computation. To apply garbled circuits to securely compute a function $f(x_1, x_2)$ in the presence of semi-honest adversaries between two parties, each party i ($i \in \{1, 2\}$) privately provides its input a_i, and then



b_i	b_j	$g(b_i,b_j)$	Garbled value
0	0	0	$\mathrm{Enc}_{(K^0_{w_i},K^0_{w_j})}(K^0_{w_k})$
0	1	0	$\mathrm{Enc}_{(K^0_{w_i},K^1_{w_j})}(K^0_{w_k})$
1	0	0	$\mathrm{Enc}_{(K^1_{w_i},K^0_{w_j})}(K^0_{w_k})$
1	1	1	$\mathrm{Enc}_{(K^1_{w_i},K^1_{w_j})}(K^1_{w_k})$

Fig. 2. A garbled AND gate

they cooperatively run a garbled circuit computing function f (represented by a boolean circuit) to obtain the output $f(a_1, a_2)$. The protocol requires that the parties do not collude with each other.

Specifically, party 1 (called garbler) generates a garbled circuit computing f. It then sends to party 2 (called evaluator) the garbled circuit, garbled a_1, and output decoding. Upon receiving all the information from the garbler, the evaluator runs a 1-out-of-2 *oblivious transfer* protocol [6][7] with the garbler, and plays the role of chooser, in order to obliviously obtain its garbled a_2. Using both garbled a_1 and a_2, the evaluator computes garbled $f(a_1, a_2)$, and then decodes it with output decoding to get $f(a_1, a_2)$ in the clear.

The core idea of garbled circuits relies in the circuit encoding. The garbler associates two random cryptographic keys, $K_{w_i}^0$ and $K_{w_i}^1$ to each wire w_i of the boolean circuit computing f that corresponds to the bit $b_i = 0$ and $b_i = 1$, respectively. For each binary gate g (e.g., AND as illustrated in Fig. 2) with input wires w_i and w_j, and output wire w_k, the garbler computes the four ciphertexts

$$\mathrm{Enc}_{(K_{w_i}^{b_i}, K_{w_j}^{b_j})}(K_{w_k}^{g(b_i, b_j)}) \text{ for } b_i, b_j \in \{0, 1\}.$$

The above four randomly ordered ciphertexts define the garbled gate g.

There are two requirements for symmetric encryption algorithm Enc used in garbled circuits:

- Enc is keyed by a pair of keys, and is indistinguishable under chosen-plaintext attacks;
- Given a key pair $(K_{w_i}^{b_i}, K_{w_j}^{b_j})$, the corresponding decryption process can unambiguously recover $K_{w_k}^{g(b_i, b_j)}$.

B. *XOR-homomorphic Encryption*

In our work, we use an XOR-homomorphic public key cryptosystem, where $E(.)$ and $D(.)$ are the encryption and decryption algorithms, satisfying XOR-homomorphism, i.e., $D(E(m) \oplus \mu) = m \oplus \mu$. That is, the XOR result of a ciphertext and a constant amounts to the ciphertext of the XOR result of the corresponding plaintext and the constant.

A XOR-homomorphic cryptosystem of semantic security is Hash-ElGamal cryptosystem [8]. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of order q, and $h : \mathbb{G} \to \{0, 1\}^l$ be a cryptographic hash function. The Hash-ElGamal cryptosystem can be described as follows.

- **Key generation**: Given the private key $k_s = x$ with $x \in \mathbb{Z}_q$, the public key is $k_p = (g, y)$, where $y = g^x$.

- **Encryption**: A message $m \in \{0,1\}^l$ is encrypted as:

$$c = (u, v) = (g^r, h(y^r) \oplus m)$$

for some random $r \in \mathbb{Z}_q$.
- **Decryption**: A ciphertext $c = (u, v)$ is decrypted by computing $m = h(u^x) \oplus v$.

It is worth remarking that one can publicly mask the ciphertext c with any chosen random mask $\mu \in \{0,1\}^l$ as $c' = (u', v') = (u, v \oplus \mu)$. Then, c' can be decrypted to yield the masked message $m' = m \oplus \mu$. Indeed, we have

$$\begin{aligned} m' &= v' \oplus h(u'^x) = (v \oplus \mu) \oplus h((g^r)^x) \\ &= ((m \oplus h(y^r)) \oplus \mu) \oplus h(y^r) = m \oplus \mu. \end{aligned}$$

C. Cryptographical Protocol Privacy

The formal definition of privacy against semi-honest adversaries in two-party computation is as follows [9].

Definition 1 (Privacy against semi-honest attackers) *Let $f(x, y)$ be a functionality with two inputs x and y, and two outputs $f_A(x, y)$ and $f_B(x, y)$. Suppose that protocol Π computes functionality $f(x, y)$ between two parties Alice and Bob. Let $V_A^\Pi(x, y)$ (resp. $V_B^\Pi(x, y)$) represent Alice's (Bob's) view during an execution of Π on (x, y). In other words, if $(x, \boldsymbol{r}_A^\Pi)$ (resp. $(y, \boldsymbol{r}_B^\Pi)$) denotes Alice's (Bob's) input and randomness, then*

$$V_A^\Pi(x, y) = (x, \boldsymbol{r}_A^\Pi, m_1, m_2, ..., m_t), \text{ and}$$
$$V_B^\Pi(x, y) = (y, \boldsymbol{r}_B^\Pi, m_1, m_2, ..., m_t),$$

where $\{m_i\}$ denote the messages passed between the parties. Let O_A^Π (resp. O_B^Π) denote Alice's (Bob's) output after an execution of Π on (x, y), and $O^\Pi(x, y) = (O_A^\Pi(x, y), O_B^\Pi(x, y))$. Then we say that protocol Π preserves privacy (or is secure) against semi-honest adversaries if there exist probabilistic polynomial time (PPT) simulators S_1 and S_2 such that

$$\{(S_1(x, f_A(x, y)), f(x, y))\} \overset{c}{\equiv} \{(V_A^\Pi(x, y), O^\Pi(x, y))\}, \quad (4)$$
$$\{(S_2(y, f_B(x, y)), f(x, y))\} \overset{c}{\equiv} \{(V_B^\Pi(x, y), O^\Pi(x, y))\}, \quad (5)$$

where $\overset{c}{\equiv}$ denotes computational indistinguishability.

In the sense of cryptography, to protect an input (privacy), the secure protocol should reveal nothing about the input exception what is revealed from the output.

IV. Our Protocol

The framework of our privacy-preserving cloud auction is illustrated in Fig. 1, where the cloud provider (CloudPdr) performs the privacy-preserving cloud auction, while a crypto-service provider (CryptPdr) enables this secure computation. Under this framework, we then design a privacy-preserving cloud auction protocol.

A. Design Rationale and Overview

To allow CloudPdr to execute the auction while either CloudPdr or CryptPdr cannot learn anything about bids other than the auction results, we apply a hybrid approach in our design by combining XOR-homomorphic encryption with garbled circuits. Specifically, our design rationale is as follows.

- First, each bidder encrypts its bid using the public key of CryptPdr, and summits the bid to CloudPdr.
- Next, CloudPdr collects the encrypted bids and cooperates with CryptPdr to share the bids using secret sharing.
- Using the boolean circuit computing the auction results, CryptPdr generates a garbled circuit and sends it to CloudPdr. How to construct the auction circuit is detailed in Section IV-B.
- CloudPdr obtains the garbled values corresponding to the inputs to the auction circuit and then executes the garbled circuit to get the auction results in the clear.

The global algorithm is illustrated in Protocol 1.

Protocol 1 Privacy-preserving Cloud Auction

Input: The boolean circuit computing auction results, the encrypted bids

Output: Auction results in the clear

Phase 1: Bidding and Bid Sharing

1: **Each bidder j:** it encrypts its bid $(\mathbf{K}_j, \mathbf{B}_j)$ using CryptPdr's public key, pk, with Hash-ElGamal encryption algorithm E. For type-i VM, it submits a pair (j, c) with $c = E_{pk}(k_j^i, b_j^i)$ to CloudPdr. Once having submitted its bid, it can go off-line.

2: **CloudPdr:** upon receiving all bids from bidders, for each encrypted part $c = E(x)$ of the bids, it chooses a random number x_1 as its share of x, and sends $c_2 = c \oplus x_1 = E(x \oplus x_1)$ (due to XOR-homomorphism of $E(\cdot)$) to CryptPdr.

Phase 2: Garbled Circuit Computing

3: **CryptPdr:** it garbles the auction circuit and its inputs (i.e. its shares of bids), and sends the garbled circuit, the garbled values of its inputs and output decoding to CloudPdr.

4: **CloudPdr and CryptPdr:** upon receiving the garbled circuit and values, and the output decoding, CloudPdr runs a 1-out-of-2 oblivious transfer protocol with CryptPdr by playing the role of the chooser so as to obliviously obtain the garbled values corresponding to its inputs (shares of bids).

5: **CloudPdr:** with the garbled values corresponding to all bid shares obtained in the last step, it computes the garbled bids using a simple XOR operation, and computes the auction results in the clear using the garbled circuit and bids, and the output decoding.

B. Main Algorithm: Computing Auction Results

As can be seen in Protocol 1, the key component in our privacy-preserving auction design is the construction of the boolean circuit computing the auction results. To design a boolean circuit executing the auction requires the auction algorithm to be *data-oblivious*, i.e., its execution path does not depend on its input. Before delving into the details of our auction algorithm, we illustrate the notion of data-obliviousnees in algorithm design.

```
int ge90(int s[], int n){
    int cnt = 0;
    for (int i = 0; i < n; ++i){
        if (s[i] >= 90){
            ++cnt;
        }
    }
    return cnt;
}
```

Programe A

```
int ge90(int s[], int n){
    int cnt = 0;
    for (int i = 0; i < n; ++i){
        int b = (s[i] >= 90);
        cnt += b;
    }
    return cnt;
}
```

Programe B

Fig. 3. Illustration of data-oblivious algorithm

Example 1 *Consider an example of designing an algorithm to compute the number of students whose scores ($s[i]$ with $i \in [0..n-1]$) are not less than 90. Two algorithms are presented in Fig. 3. The execution path of Program A depends on its input $s[i]$ and thus is not data-oblivious, while the execution path of Program B remains the same regardless of its input, and is thus data-oblivious.*

Our data-oblivious algorithm design for the cloud auction presented in Section II-B is illustrated in Alg. 2. Note that, throughout this algorithm, all basic operations, such as comparison, swap, addition, multiplication, division of two integers, square root of an integer, and sorting should be data-oblivious. To address this design challenge, we develop data-oblivious implementation of these operations in Section IV-C.

Algorithm 2 Data-oblivious Cloud Auction

Input: $\mathbf{K}$, $\mathbf{B}$, and weight vector $\mathbf{W} = \{w_i\}_m$
Output: $\mathbf{X} = \{x_j\}_n$, $\mathbf{P}$, $\{x_j\mathbf{K}_j\}_{j=1}^n$

```
Bid Ranking:
1:  for $j = 1..n$ do
2:      $b_j \leftarrow \sum_{i=1}^m k_j^i b_j^i$; $\omega_j \leftarrow \sqrt{\sum_{i=1}^m k_j^i w_i}$; $b_j^* \leftarrow \frac{b_j}{\omega_j}$
3:  end for
4:  Sorts bidders in descending order of $b_j^*$, and abuses same indexes
    after sorting;
VM Allocation:
5:  $\mathbf{X} \leftarrow \{0\}_n$; $\{s_i\}_{i=1}^m \leftarrow \{0\}_m$
6:  for $j = 1..n$ do
7:      $x_j \leftarrow \wedge_{i=1}^m [s_i + k_j^i \le k_i]$
8:      $\{s_i\}_{i=1}^m \leftarrow \{s_i + k_j^i x_j\}_{i=1}^m$
9:  end for
Pricing:
10: $\mathbf{P} \leftarrow \{0\}_n$; $\{s_i\}_{i=1}^m \leftarrow \{0\}_m$
11: for $j = 1..n$ do
12:     $\{t_i\}_{i=1}^m \leftarrow \{s_i\}_{i=1}^m$; $\{\delta_u\}_{u=j}^n \leftarrow \{0\}_{n-j+1}$
13:     $\{\lambda_u\}_{u=j}^n \leftarrow \{0\}_{n-j+1}$; $\{\theta_u\}_{u=j}^n \leftarrow \{0\}_{n-j+1}$
14:     for $u = j+1..n$ do
15:         $\delta_u \leftarrow \wedge_{i=1}^m [t_i + k_u^i \le k_i]$
16:         $\{t_i\}_{i=1}^m \leftarrow \{t_i + \delta_u k_u^i\}_{i=1}^m$
17:         $\lambda_u \leftarrow \vee_{i=1}^m [t_i + k_j^i > k_i]$
18:         $\theta_u \leftarrow \lambda_{u-1} \oplus \lambda_u$
19:         $p_j \leftarrow p_j + x_j \theta_u b_u^* \omega_j$
20:     end for
21:     $\{s_i\}_{i=1}^m \leftarrow \{s_i + k_j^i x_j\}_{i=1}^m$;
22: end for
23: Sorts bidders in ascending order of $id_j$, and abuses same indexes
    after sorting;
        return $\mathbf{X}$, $\mathbf{P}$, $\{x_j\mathbf{K}_j\}_{j=1}^n$;
```

1) Bid Ranking: This step computes b_j^* for each bidder j, and sorts the bidders in descending order of b_j^*.

- Firstly, the "for" loop computes b_j, ω_j and then b_j^* for each bidder j (Lines 1 to 3), which is straightforward with all needed basic operations implemented data-obliviously.
- Secondly, bidders are sorted data-obliviously in descending order of b_j^* (Line 4), and VM instances will be allocated to bidders in this order.

2) VM Allocation: This step allocates VM instances to each winner in the order sorted previously. Variable s_i denotes the number of allocated type-i VM instances (Lines 5 & 8), and binary flag x_j indicates whether the current remaining VM resources can satisfy bidder j's request (Lines 5 & 7), that is, whether Equation (2) is satisfied ($x_j = 1$) or not ($x_j = 0$). The allocation computation is expressed using x_j, which also indicates the allocation result for bidder j.

3) Pricing: This step, computing the price charged to each bidder, is the most challenging part. Since the original pricing process (cf. Alg. 1 in [3]) is not data-oblivious due to the numerous "if" branches, to achieve data-obliviousness, the algorithm needs to exhibit the same execution path regardless of bid values to find the critical values of bidders.

To address the challenge, our main idea is to introduce appropriate binary flags indicating different conditions, and then use them to data-obliviously compute critical bidders. Specifically, to charge each bidder j (Line 11), we reallocate VMs to all bidders except bidder j (Line 14. Note that since bidders before j are allocated exactly the same regardless of j's presence, we only need to reallocate VMs to bidders starting from $j+1$ if we store and use previous states, cf. Lines 12 & 21.), and define three arrays of binary flags, δ_u, λ_u and θ_u with $u \in [j..n]$, as follows:

δ_u**:** indicates whether bidder u is served ($\delta_u = 1$) or not ($\delta_u = 0$) if bidder j is absent (Lines 12 & 15).

λ_u**:** indicates whether bidder j is denied ($\lambda_u = 1$) or not ($\lambda_u = 0$) if it is placed after bidder u (Lines 13 & 17).

θ_u**:** indicates whether bidder u is j's critical bidder ($\theta_u = 1$) or not ($\theta_u = 0$) (Lines 13 & 18).

For $u = j$, obviously $\delta_j = \lambda_j = \theta_j = 0$. According to its definition, j's critical bidder (bidder c) should be served if bidder j is absent, i.e. $\delta_c = 1$; also, bidder j should be denied if it is placed after c, i.e. $\lambda_c = 1$. Further analysis shows that once bidder j is denied after bidder c, it will always be denied afterwards, and if it is placed before bidder c, it will always be served (otherwise, bidder c cannot be the critical bidder of bidder j, which leads to contradiction). Therefore, the values of $\{\lambda_u\}_{u=j}^n$ always follows the pattern $00\cdots011\cdots1$ where we denote the first bit 1 as λ_c. The three flag arrays can be written as follows:

$$\begin{pmatrix} u: & j & j+1 & \ldots & c-1 & c & c+1 & \ldots & n \\ \delta_u: & 0 & \delta_{j+1} & \ldots & \delta_{c-1} & 1 & \delta_{c+1} & \ldots & \delta_n \\ \lambda_u: & 0 & 0 & \ldots & 0 & 1 & 1 & \ldots & 1 \\ \theta_u: & 0 & 0 & \ldots & 0 & 1 & 0 & \ldots & 0 \end{pmatrix}$$

Using the above analysis, we can easily find the critical bidder c of bidder j by computing (Line 18)

$$\theta_u = \delta_u(\lambda_{u-1} \oplus \lambda_u) = \lambda_{u-1} \oplus \lambda_u.$$

We then compute j's price using flags x_j and θ_u (Line 19).

4) Discussion and Remarks: There are some technical remarks worth discussing.

First, when sorting bidders, we mean to sort an array of tuples, each of which is of the form $(x_j, id_j, p_j, b_j^*, \omega_j, \mathbf{K}_j)$ representing a bidder. The same indexes are abused for convenience after each sorting.

Secondly, the second sorting in Line 23 is indispensable. Otherwise, the protocol cannot preserve privacy. The reason is as follows. Without this sorting, the bidder tuples will still remain in descending order of b_j^*. Then, when the auction results, including IDs of winners, are revealed, the order of winners in term of b_j^* is also leaked. Such information about bids cannot be inferred from the auction results, hence privacy is no more preserved. In contrast, through the sorting, the order of winners are independent of bid values (assuming that ids are independent of bid values), information leakage is thus avoided.

Thirdly, in many practical scenarios, the number of bidders n is much larger than the number of VM types m. In such context, the computational complexity of Alg. 2 is $O(n^2)$. Later in Section V, we further present an improved algorithm of computational complexity $O(n \log^2 n)$ when n is larger.

C. Data-oblivious Building-block Operations

As stated previously, all basic operations in the main algorithm (Alg. 2) need to be implemented data-obliviously. These operations include integer comparison, swapping, integer addition, subtraction, multiplication, division and square root, and sorting network. We now design data-oblivious algorithms (circuits) for these operations. We refer to them data-oblivious building-block algorithms, or building-block algorithms in brief.

Throughout this subsection, we denote two K-bit integers by $x = (x_K x_{K-1} ... x_2 x_1)$ and $y = (y_K y_{K-1} ... y_2 y_1)$. Our design of building-block algorithms is presented as follows with their pseudo-codes illustrated in Alg. 3.

- **Integer comparison.** We apply the circuit proposed in [10] for integer comparison (cf Alg. 3(a)), which is optimized by "free-XOR" gates. Specifically,
  - for $c_1 = 0$, the comparison result $c_{K+1} = [x > y]$;
  - for $c_1 = 1$, $c_{K+1} = [x \geq y]$.

  The comparison circuits for $[x < y]$ and $[x \leq y]$ can be obtained by switching x and y. In terms of implementation complexity, the K-bit integer comparison circuit only contains K NON-XOR gates.
- **Swapping and multiplexing.** Suppose that x and y are the two bit strings to swap, and the swap indicator is denoted by b such that if $b = 1$, swap x and y; else, remain untouched. The swapping algorithm (cf Alg. 3(b)) can be optimized with only K NON-XOR gates [8]. In the algorithm, $x' = (x'_K x'_{K-1} \cdots x'_2 x'_1)$ and $y' = (y'_K y'_{K-1} \cdots y'_2 y'_1)$ are the swapping results. If the swapping algorithm only returns x', then we get a multiplexer $\mathtt{mux}(x, y, b)$, with $\mathtt{mux}(x, y, 0) = x$, and $\mathtt{mux}(x, y, 1) = y$.
- **Integer Addition and Subtraction.** Integer addition (cf Alg. 3(c)) with two integers x and y can be optimized with only K NON-XOR gates [10]. The two integers can be unsigned integers or signed integers using two's complement representation, where subtraction of two integers $x - y$ can be computed by addition $x + (-y)$. The algorithm `subAbs` (cf Alg. 3(d)) finds absolute difference of two unsigned integers x and y. In `subAbs`, `twoCompRep` inverts all bits of the input, adds one and returns the result; `abs` returns the absolute value of the input.
- **Integer Multiplication.** Our design is based on Karatsuba multiplication [11], a fast algorithm for integer multiplication with computational complexity of $O(K^{1.6})$. In `mul` (cf Alg. 3(e)), x and y are two unsigned integers, and $K = 2^k$. The extension to general K is straightforward by padding 0 preceding x_K and y_K.
- **Integer Division.** The data-oblivious algorithm for integer division (cf Alg. 3(f)) with two unsigned integers x and y can be designed using "school method".
- **Integer Square Root.** We design the data-oblivious algorithm for integer square root (cf Alg. 3(g)) of an unsigned integer x using bit-by-bit computation [12].
- **Sorting Network.** In our main algorithm, we need to sort the bidders in a certain way. To implement data-oblivious sorting, we employ an odd-even merge sorting network [13], [8], a circuit that sorts an input sequence $(a_1, a_2, ..., a_n)$ into a monotonically increasing sequence $(a'_1, a'_2, ..., a'_n)$. The main component of sorting network is the compare-and-swap circuits, a binary operator taking as input a pair (a_1, a_2), and returning the sorted pair a'_1, a'_2, with $a'_1 = \min(a_1, a_2)$ and $a'_2 = \max(a_1, a_2)$. As desirable properties, the odd-even merge sorting network is data-oblivious by nature and achieves a computation complexity of $O(n \log^2 n)$. In practice, it outperforms most widely used sorting network algorithms [14]. The algorithms `oeSort` and `oeMerge` (cf Alg. 3(h) and (i)) illustrate the odd-even merge sorting for any $n \geq 0$.

D. Security Analysis

In this subsection, we prove that our protocol preserves privacy in the sense of cryptography.

Theorem 1 *As long as CloudPdr and CryptPdr do not collude, Protocol 1 (using Alg. 2) preserves privacy against semi-honest adversaries.*

Proof: We prove the privacy preservation by distinguishing two phases: (1) bidding and bid sharing, and (2) garbled circuit computation:

In the bidding and bid sharing phase, each bidder j encrypts its bid $(\mathbf{K}_j, \mathbf{B}_j)$ using Hash-ElGamal encryption, and sends the ciphertexts to CloudPdr. Because Hash-ElGamal system is

Algorithm 3 Data-oblivious building-block algorithms

(a) Integer Comparison: comp(x, y, c_1)

```
1: for $i = 1..K$ do
2:     $c_{i+1} \leftarrow x_i \oplus ((x_i \oplus c_i) \wedge (y_i \oplus c_i))$
3: end for
       return $c_{K+1}$
```

(b) Swapping: swap(x, y, b)

```
1: for $i = 1..K$ do
2:     $x'_i \leftarrow [b \wedge (x_i \oplus y_i)] \oplus x_i$; $y'_i \leftarrow x'_i \oplus (x_i \oplus y_i)$
3: end for
       return $x' = (x'_K x'_{K-1} \cdots x'_1)$, $y' = (y'_K y'_{K-1} \cdots y'_1)$
```

(c) Integer Addition: add(x, y)

```
1: $c_1 = 0$
2: for $i = 1..K$ do
3:     $s_i \leftarrow x_i \oplus y_i \oplus c_i$; $c_{i+1} \leftarrow c_i \oplus ((x_i \oplus c_i) \wedge (y_i \oplus c_i))$
4: end for
       return $s = (c_{K+1} s_K \cdots s_2 s_1)$
```

(d) Integer Subtraction: subAbs(x, y)

```
1: $xx \leftarrow$twoCompRep$(x)$; $yy \leftarrow$twoCompRep$(-y)$
2: $ss \leftarrow$add$(xx, yy)$; $s \leftarrow$abs$(ss)$
       return $s$
```

(e) Integer Multiplication: mul(x, y)

```
1: if $K = 1$ then
2:     $p \leftarrow x \wedge y$
3: else
4:     $u_1 \leftarrow (x_K x_{K-1} \cdots x_{\frac{K}{2}+1})$; $u_0 \leftarrow (x_{\frac{K}{2}} x_{\frac{K}{2}-1} \cdots x_1)$
5:     $v_1 \leftarrow (y_K y_{K-1} \cdots y_{\frac{K}{2}+1})$; $v_0 \leftarrow (y_{\frac{K}{2}} y_{\frac{K}{2}-1} \cdots y_1)$
6:     $p_0 \leftarrow$mul$(u_0, v_0)$; $p_2 \leftarrow$mul$(u_1, v_1)$
7:     $p_1 \leftarrow$mul$(u_1 - u_0, v_0 - v_1)$ // Need sign treatment
8:     $p \leftarrow (2^K + 2^{\frac{K}{2}})p_2 + 2^{\frac{K}{2}} p_1 + (2^{\frac{K}{2}} + 1)p_0$
9: end if
       return $p$
```

(f) Integer Division: div(x, y)

```
1: for $i = K..1$ do
2:     $yy \leftarrow (y << (i-1))$
3:     $q_i \leftarrow$comp$(x, yy, 1)$;//$x \geq yy$
4:     $xx \leftarrow$mux$(x$, subAbs$(x, yy)$, $q_i)$
5: end for
       return $q = \lfloor \frac{x}{y} \rfloor = (q_K q_{K-1} \cdots q_1)$
```

(g) Integer Square Root: sqrt(x)

```
1: $f \leftarrow 0$
2: for $i = K..1$ do
3:     if $(i > \lceil \frac{K}{2} \rceil)$ then
4:         $r_i \leftarrow 0$
5:     else
6:         $e \leftarrow (1 << 2(i-1))$
7:         $r_i \leftarrow$comp$(x, f + e, 1)$
8:         $x \leftarrow$mux$(x, (x - f - e), r_i)$
9:         $f \leftarrow$mux$(f >> 1, (f >> 1) + e, r_i)$
10:    end if
11: end for
       return $r = \lfloor \sqrt{x} \rfloor = (r_K r_{K-1} \cdots r_1)$
```

(h) Odd-even Merge Sorting: oeSort(a, inc)

```
Input: $a = \{a_1, a_2, \cdots, a_n\}$, $inc = 0$ or 1
Output: $c$ ($inc = 0$: non-increasing, $inc = 1$: non-decreasing)
1: if $n > 1$ then
2:     $m \leftarrow \frac{n+1}{2}$;
3:     oeSort$(a[1..m], inc)$; oeSort$(a[m+1..n], inc)$;
4:     $c \leftarrow$oeMerge$(a[1..m], a[m+1..n], inc)$;
5: end if
       return $c$;
```

(i) oeMerge(a, b, inc)

```
Input: $a = \{a_1, a_2, \cdots, a_m\}$, $b = \{b_1, b_2, \cdots, b_n\}$, $inc = 0$ or 1
Output: $c$ ($inc = 0$: non-increasing, $inc = 1$: non-decreasing)
1: if $(m = 0 \,||\, n = 0)$ then
2:     $c \leftarrow \langle a, b \rangle$;
3: else if $(m = 1$ && $n = 1)$ then
4:     swap$(a_1, b_1, inc = 1$ ? $[a_1 > b_1]$ : $[a_1 < b_1])$;
5:     $c \leftarrow \langle a, b \rangle$;
6: else
7:     $c[1, 3, \cdots] \leftarrow$oeMerge$(a[1, 3, \cdots], b[1, 3, \cdots], inc)$;
8:     $c[2, 4, \cdots] \leftarrow$oeMerge$(a[2, 4, \cdots], b[2, 4, \cdots], inc)$;
9:     for $(i = 2;\ i < n;\ i \leftarrow i + 2)$ do
10:        $bl \leftarrow (inc = 1)$ ? $[c[i] > c[i+1]]$ : $[c[i] < c[i+1]]$;
11:        swap$(c[i], c[i+1], bl)$;
12:    end for
13: end if
       return $c$;
```

semantically secure, CloudPdr can know nothing about the bids. Then, CloudPdr chooses a random number for each bid cipertext and XOR the cipertext with this number, and sends the XORed ciphertexts to CryptPdr. According to the XOR-homomorphism of Hash-ElGamal encryption, this XOR operation is in fact equivalent to XORing the plain bid values with random numbers. Thus, although CryptPdr can decrypt the ciphertexts and get the XORed bids, it knows nothing about the bid values provided that the random numbers are not leaked to it from CloudPdr. Therefore, as long as CloudPdr and CryptPdr do not collude with each other, the bidding and bid sharing phase preserves privacy of bidders.

In the garbled circuit computation phase, the main algorithm (Alg. 2) calls the building-block algorithms to build a boolean circuit that computes the auction results. Based on the boolean circuit, a garbled circuit is then constructed and computed between CryptPdr and CloudPdr, with the former being the garbler and the latter being the evaluator. The protocol follows exactly Yao's protocol, whose security (privacy) was proved theoretically in [5], and the protocol output is defined exactly as the auction results. Therefore, neither CryptPdr nor CloudPdr knows anything about the bids beyond the auction results, as long as they do not collude with each other.

Since the two phases are composed sequentially, it follows from the sequential composition theory [15] that Protocol 1 preserves privacy against semi-honest adversaries. □

V. An Improved Protocol

Though Protocol 1 achieves provable privacy-preservation against semi-honest adversaries, its computational complexity is $O(n^2)$ regardless of the number of winners. In many practical scenarios, resources are constrained and the number of bidders n is much larger than the number of winners W, i.e., $n \gg W$. Additionally, it usually holds $n \gg m$ and $n \gg K$, where $K = \max_i k_i$. Under this context, the cloud auction in the clear presented in Sec II has a computational complexity of $O(n \log n)$, which scales more efficiently than the privacy-preserving auction presented in Protocol 1. In this section, we narrow the performance gap by developing an improved protocol with computational complexity $O(n \log^2 n)$, i.e. within only a logarithmic factor of the cloud auction in the clear.

A. A Naive Idea

A naive idea to improve our protocol is to let the protocol reveal x_j's before the pricing step in Alg. 2, and then with these x_j's, the protocol only prices a bidder j with $x_j = 1$, while setting the price to 0 for bidders with $x_j = 0$. In this way, we can achieve a computational complexity of $O(n \log^2 n)$.

However, this naive idea fails to preserve privacy. The reason is as follows. At the point of revealing x_j in Alg. 2, the bidder tuples are in descending order of b_j^*, and this order should be maintained so as to do the pricing. Consequently, both CloudPdr and CryptPdr will learn not only the auction winners, but also the order of the winners in terms of b_j^*, which is beyond the auction results. Such information leakage makes the naive idea fail to preserve privacy.

B. The Improved Algorithm

We now present our improved algorithm, which is shown in Alg. 4. There are two major improvements in our design.

- In the VM allocation, the improved protocol calculates the number of winners (W), and reveals this number to both CloudPdr and CryptPdr (Lines 5 & 8). Note that W can be revealed from the auction results, hence revealing W does not leak information.
- In the pricing, a copy of tuple array is sorted in descending order of x_j and id_j (Line 9), so that the winner tuples are placed in the first W positions and their order is independent of bids (in term of id's). Then, the first W winner tuples are cut out, and the improved protocol computes the prices of the winners by finding their critical bidders. Note that here indexes $\{j_h\}_{h=1}^W$ of winners (in the original array of tuples sorted in descending order of b_j^*) are only used to index these winner tuples, CloudPdr does not actually know them (Lines 11, 15, 17 & 19). Therefore, we need to search the critical bidder for each winner from the start to the end of the tuple array, as shown by the inner for loop (Lines 14 to 20). Specifically, we define three similar binary flag arrays δ_u, λ_u and θ_u with $u \in [0..n]$. The indications are the same as those in Section IV-B except that we replace bidder j by bidder j_h. Similar analysis can show that the critical bidder of bidder j_h can be computed by $\theta_u = \lambda_{u-1} \oplus \lambda_u$. The prices of all winners can further be computed.

Concretely, as illustrated in Alg. 4, the number of winners W is revealed to both CloudPdr and CryptPdr before pricing.

Algorithm 4 Data-oblivious Cloud Auction: Improved Algorithm

```
Input: K, B, and weight vector $\mathbf{W} = \{w_i\}_m$
Output: $\{id_j | x_j = 1\}$, $\{p_j | x_j = 1\}$, $\{\mathbf{K}_j | x_j = 1\}$
Bid Ranking:
 1: Same as Alg. 2.
VM Allocation:
 2: $\mathbf{X} \leftarrow \{0\}_n$; $\{s_i\}_m \leftarrow \{0\}_m$; $W \leftarrow 0$
 3: for $j = 1..n$ do
 4:     $x_j \leftarrow \wedge_{i=1}^m [s_i + k_j^i \le k_i]$
 5:     $W \leftarrow W + x_j$
 6:     $\{s_i\}_m \leftarrow \{s_i + k_j^i x_j\}_m$
 7: end for
 8: Reveals $W$ cooperatively.
Pricing:
 9: Sorts a copy of the tuple array in descending order of $x_j$ and $id_j$,
    and the first $W$ bidders in this copy (the winners) are indexed
    by $\{j_h\}_{h=1}^W$ in the original array before sorting.
10: $\mathbf{P} \leftarrow \{0\}_n$
11: for $h = 1..W$ do
12:     $\{t_i\}_{i=1}^m \leftarrow \{0\}_m$; $\{\delta_u\}_{u=0}^n \leftarrow \{0\}_{n+1}$
13:     $\{\lambda_u\}_{u=0}^n \leftarrow \{0\}_{n+1}$; $\{\theta_u\}_{u=0}^n \leftarrow \{0\}_{n+1}$
14:     for $u = 1..n$ do
15:         $\delta_u \leftarrow (\wedge_{i=1}^m [t_i + k_u^i \le k_i]) \wedge [id_{j_h} \ne id_u]$;
16:         $\{t_i\}_{i=1}^m \leftarrow \{t_i + \delta_u k_u^i\}_{i=1}^m$;
17:         $\lambda_u \leftarrow \vee_{i=1}^m [t_i + k_{j_h}^i > k_i]$;
18:         $\theta_u \leftarrow \lambda_{u-1} \oplus \lambda_u$;
19:         $p_{j_h} \leftarrow p_{j_h} + \theta_u b_u^* \omega_{j_h}$;
20:     end for
21: end for
        return $\{id_{j_h}\}_{h=1}^W$, $\{p_{j_h}\}_{h=1}^W$, $\{\mathbf{K}_{j_h}\}_{h=1}^W$;
```

It can be straightforwardly proved that when $n \gg m$, $n \gg K$, and $n \gg W$, the computational complexity of Alg. 4 is $O(n \log^2 n)$.

C. Security Analysis

In this subsection, we prove that the improved protocol achieves privacy-preservation.

Theorem 2 *As long as CloudPdr and CryptPdr do not collude, the improved protocol (Protocol 1 using Alg. 4) preserves privacy against semi-honest adversaries.*

Proof: We prove the privacy preservation of the improved protocol by distinguishing two phases: (1) bidding and bid sharing, and (2) garbled circuit computation. The first part remains the same as that when using Alg. 2 and can be proved using the same argument as in Theorem 1. We now prove that the second phase also preserves privacy.

The second phase of garbled circuit computation in Alg. 4 is sequentially composed of two garbled circuits:

- The first circuit corresponds to the bid ranking and VM allocation phases in the algorithm. This circuit computes the allocation results, and publishes the number of winners.
- The second circuit corresponds to the pricing phase in the algorithm. This circuit uses the allocation results and the number of winners computed by the first circuit to compute the auction results.

It follows from the security of garbled circuits and the sequential composition theory that the two circuits only reveal the auction results, and the number of winners. However, the latter can be inferred from the auction results. We thus conclude that the improved protocol (using Alg. 4) preserves privacy against semi-honest adversaries. □

VI. Experiments

We implement both our original protocol (Protocol 1 using Alg. 2) and our improved protocol (Protocol 1 using Alg. 4) for cloud auctions on top of FastGC [16], a Java-based framework for garbled circuit computation. In our experiments, we simulate CloudPdr and CryptPdr with two processes on a computer.

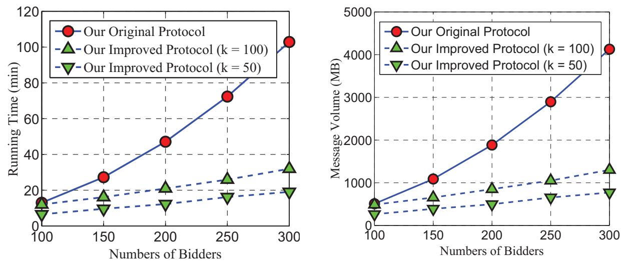
Experimental settings are as follows: Per-instance bids, and the numbers of instances are generated randomly in the intervals $[0..100]$ and $[0..3]$, respectively; The number of VM types m is set to 6 by default; The number of each VM type i's instances possessed by CloudPdr k_i is set to k. Numbers are represented using 16 bits. Throughout our simulations, each point represents the average value of a number of independent simulation runs, with the required number of simulation runs calculated using "independent replications" [17]. In our simulation, we focus on the following performance metrics:

- Computation overhead: total CPU time spent by CloudPdr and CryptPdr,
- Communication overhead: total message volume (data size of all messages sent between the two parties).

Fig. 4 traces the experimental results of our original protocol and our improved protocol as the number of bidders increases from 100 to 300, with $k = 100$ and $k = 50$, respectively. From the results, we make the following observations:

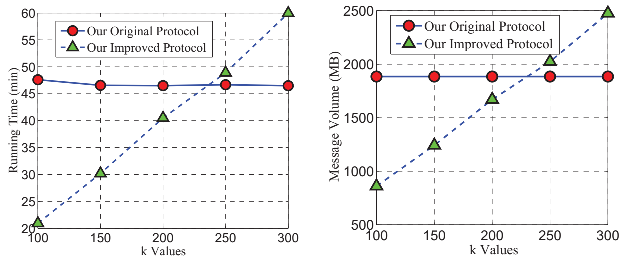
- *Curve Trend.* Both the computation and communication overhead of our original protocol grow super-linearly in the number of bidders n, while these curves of our improved protocol grow quasi-linearly, which is in accordance with the theoretical results.
- *Performance Comparison.* Given the number of bidders, the computation and communication overhead of our original protocol are larger than those of our improved protocol. The gap becomes more significant as the number of bidders increases, and as k decreases. For example, when the number of bidders is 250, the running time and message volume of our original protocol are 72 min and 2898 MB, respectively, and those of our improved protocol are 26 min and 1051 MB for $k = 100$, and are 16 min and 654 MB for $k = 50$.

Fig. 5 shows the performance comparison between our original protocol and our improved protocol as the computing resources changes, that is, the value k varies from 100 to 300, with the number of bidders fixed to 200. From the figure, we observe that our improved protocol does not always outperform our original protocol. When k is relatively large, i.e., there are sufficient computing resources, our improved protocol is outperformed by our original protocol due to its additional task of finding critical bidder for each winner (which searches for all bidders). Therefore, as stated in Section V, our improved protocol can bring performance gain when there are a large number of bidders compared to constrained computing resources.



(a) Computation Overhead (b) Communication Overhead

Fig. 4. Computation and communication comparisons for our protocols as the number of bidders n varies



(a) Computation Overhead (b) Communication Overhead

Fig. 5. Computation and communication comparisons for our protocols as k varies

Tab. II further illustrates the performance of our improved protocol when the computing resources are constrained. We can see that the computation and communication overhead are acceptable in practical scenarios even when there are a large number of bidders.

VII. Related Work

In this section, we review existing work on secure and privacy-preserving auctions and mechanism design and discuss specific challenges in the privacy-preserving cloud auction design guiding our design methodology.

There exist a number of privacy-preserving designs for conventional auctions, such as [18], [19], [20], [21]. However, most of these studies [19], [20], [21] did not preserve privacy in the sense of cryptography, i.e., they leaked some information about bids beyond auction results (not necessary the exact bid values). The study [18] did preserve privacy in the sense of cryptography as our work, but it provided a secure auction framework without any concrete secure auction design. Furthermore, all these studies did not implement the

TABLE II
Computation and Communication Performance for Large n and Fixed $k = 100$

Number of Bidders	200	400	600	800	1000
Run Time (min)	23	44	71	93	115
Commu. Data (MB)	910	1793	2888	3768	4661

secure auctions and did not consider practical efficiency issues. Instead, our work focuses on both privacy preservation in the sense of cryptography and the practical efficiency of auction. Also, our work is motivated by the challenges posed by the specifical context of cloud auction. Such specific constraints make our problem non-trivial that cannot build upon any existing work, thus calling for an original study.

A related research topic that we need to mention is the secure spectrum auction (cf.[22], [23], [24], [25], [26], [27], [28] etc.). Among this strand of research, previous studies mainly focused on designing secure auctions that enable spectrum reusability, few of them considered heterogeneity of resources. The two studies [22], [23] did take into consideration spectrum heterogeneity, but they failed to provide security in the sense of cryptography [25]. In our work, we design privacy-preserving auction protocols for heterogeneous cloud computing resources, with the aim of preserving privacy in the sense of cryptography and achieving practical computation and communication efficiency.

Another concept that is orthogonal to ours is differential privacy [29], [30], [31], which guarantees the insensibility of the output on the input by adding noise. However, differential privacy does not ensure security in the sense of cryptography, which is the objective of our work. As an orthogonal component, differential privacy can be applied to further enhance the security in our solution to prevent an attacker from inferring individual winning bidder information from the auction output.

VIII. Conclusion

In this paper, we have proposed a privacy-preserving framework for truthful cloud auctions. Technically, our design of a privacy-preserving cloud auction addresses the specific challenges posed by the cloud auction context by leveraging the techniques in garbled circuits and homomorphic encryption. Our protocol preserves privacy in the sense of cryptography, such that it reveals nothing about the bids to any auction participant and the external attacker, except the auction results. We have further improved our protocol, such that its complexity is within a logarithmic factor of the cloud auction in the clear while its security remains unchanged. We have also implemented the our proposition in Java, and theoretically and experimentally shown that it preserves privacy by incurring only limited computation and communication overhead.

Acknowledgment

The work is supported by the Natural Science Foundation of China under Grant Nos. 61572031, 61572001 & 61202407, and the Natural Science Foundation of Anhui Province under Grant No. 201508085QF132.

References

[1] W. Shi, C. Wu, and Z. Li. RSMOA: A revenue and social welfare maximizing online auction for dynamic cloud resource provisioning. In *Proc. IEEE/ACM IWQoS*, 2014.

[2] L. Zhang, Z. Li, and C. Wu. Dynamic resource provisioning in cloud computing: A randomized auction approach. In *Proc. IEEE INFOCOM 2014*, pages 433–441.

[3] Q. Wang, K. Ren, and X. Meng. When cloud meets ebay: Towards effective pricing for cloud computing. In *Proc. Infocom*, 2012.

[4] A. C.-C. Yao. How to generate and exchange secrets. In *Proc. FOCS*, 1986.

[5] Y. Lindell and B. Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2), 2009.

[6] M. O. Rabin. How to exchange secrets by oblivious transfer. Technical report, Harvard University, 1981.

[7] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[8] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh. Privacy-preserving matrix factorization. In *Proc. ACM CCS*, 2013.

[9] O. Goldreich. *Foundations of Cryptography: Volume 2-Basic Applications*. Cambridge University Press, 2004.

[10] V. Kolesnikov, A. R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Proc. CANS*, 2009.

[11] D. E. Knuth. *The Art of Computer Programming, Vol. 2, Semuninumerical Algorithms, 3rd ed.* Addison-Wesley, 1997.

[12] J. W. Crenshaw. Integer square roots. *Embedded Systems Programming*, 1998.

[13] K. E. Batcher. Sorting networks and their applications. In *Proc. AFIPS Spring Joint Computer Conference*, 1968.

[14] G. Wang, T. Luo, M. T. Goodrich, W. Du, and Z. Zhu. Bureaucratic protocols for secure two-party sorting, selection, and permuting. In *Proc. AsiaCCS*, 2010.

[15] C. Hazay and Y. Lindell. *Efficient secure two-party protocols: Techniques and constructions*. Springer, 2010.

[16] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *Proc. USENIX Security*, 2011.

[17] W. Whitt. The efficiency of one long run versus independent replications in steady-state simulation. *Management Science*, 37(6):645–666, 1991.

[18] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proc. EC*, 1999.

[19] K. Peng, C. Boyd, E. Dawson, and K. Viswanathan. Robust, privacy protecting and publicly verifiable sealed-bid auction. In *Proc. ICICS*, 2002.

[20] K. Suzuki and M. Yokoo. Secure generalized vickrey auction using homomorphic encryption. In *Proc. FC*, 2003.

[21] M. Yokoo and K. Suzuki. Secure generalized vickrey auction without third-party servers. In *Proc. FC*, 2004.

[22] M. Pan, J. Sun, and Y. Fang. Purging the back-room dealing: Secure spectrum auction leveraging paillier cryptosystem. *IEEE JSAC*, 29(4):866–876, 2011.

[23] S. Liu, H. Zhu, R. Du, C. Chen, and X Guan. Location privacy preserving dynamic spectrum auction in cognitive radio network. In *Proc. of ICDCS*, pages 256–265, 2013.

[24] Q. Huang, Y. Tao, and F. Wu. SPRING: A strategy-proof and privacy preserving spectrum auction mechanism. In *Proc. INFOCOM*, 2013.

[25] Z. Chen, L. Huang, L. Li, W. Yang, H. Miao, M. Tian, and F. Wang. PS-TRUST: Provably secure solution for truthful double spectrum auctions. In *Proc. INFOCOM*, 2014.

[26] H. Huang, X. Li, Y. Sun, and H. Xu. PPS: Privacy-preserving strategyproof social-efficient spectrum auction mechanisms. *IEEE Transactions on Parallel and Distributed Systems*, 26(5):1393–1404, 2014.

[27] Z. Chen, L. Huang, and L. Chen. ITSEC: An information-theoretically secure framework for truthful spectrum auctions. In *Proc. of INFOCOM*, pages 2065–2073, 2015.

[28] Q. Huang, Y. Gui, F. Wu, and G. Chen. A general privacy-preserving auction mechanism for secondary spectrum markets. *IEEE/ACM Transactions on Networking*, 2015.

[29] C. Dwork. Differential privacy. In *Proc. ICALP*, 2006.

[30] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Proc. FOCS*, 2007.

[31] R. Zhu, Z. Li, F. Wu, K. Shin, and G. Chen. Differentially private spectrum auction with approximate revenue maximization. In *Proc. of MobiHoc*, pages 185–194, 2014.