

A SECURE AUCTION SERVICE

KIM POTTER KIHLSSTROM

Department of Mathematics and Computer Science
Westmont College, 955 La Paz Road, Santa Barbara, CA 93108
kimkihls@westmont.edu

N. NARASIMHAN, L. E. MOSER and P. M. MELLIAR-SMITH

Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
{*nitya, moser, pmms*}@alpha.ece.ucsb.edu

ABSTRACT

In this paper we describe a distributed “English” auction service that remains correct and reliable despite the malicious corruption of some of the bidders and the auctioneer. We describe the properties of such an auction service, the implementation of the auction service, and how the auction service satisfies the required properties. The auctioneer is replicated for fault tolerance, and all communication is performed using a secure group communication system. The properties provided by the secure group communication system ensure that malicious bidders and auctioneers are not able to corrupt the integrity and security of the auction, even if some of the auctioneer replicas conspire together maliciously.

KEY WORDS

distributed systems, reliability, fault tolerance, security, electronic commerce, English auction, Byzantine faults

1. Introduction

In our technological society, many activities that previously employed face-to-face methods and paper documentation can now be replaced with electronic equivalents. For example, digital libraries can substitute for physical collections of books or journals, and courses can be taken on-line instead of in a classroom. On-line auction services have become a very popular alternative to in-person auctions. However, on-line auctions require a certain level of trust. Bidders in an on-line auction must have confidence that the auction service is honest, and the auction service needs to know that bids are authentic.

We describe an ascending bid, open cry (English) auction service that remains correct and reliable despite the malicious behavior of some of the bidders and auctioneers. We define the properties that must be satisfied in such an auction service, describe the implementation of the auction service, and discuss how the auction application provides the required properties.

2. Auctions

There are a number of different types of auctions. Bids may be sealed or open, and bid prices may proceed in ascending or descending order. A number of researchers have studied sealed bid electronic auctions, *e.g.*, [1, 2, 3, 4, 5, 6]. Prior work has also addressed open bid electronic auctions, *e.g.*, [7, 8, 9]. We consider an open bid, ascending price (English) auction. In such an on-line auction, it is important to ensure fairness to the bidders. For example, bidders need confidence that they are receiving accurate information regarding the bids submitted, and that all bids are treated fairly. The properties required for an English auction include:

- **Inclusion:** all valid bids (*i.e.*, bids for an amount greater than the highest bid to date) submitted during the auction period are included
- **Exclusion:** no bids submitted outside the auction period are included
- **Uniform receipt:** all bidders receive the same auction information
- **Ordering:** bids are processed in the order in which they are submitted by the bidders
- **Authentication:** bidders cannot masquerade as other bidders
- **Non-repudiation:** bidders cannot deny having submitted a bid.

3. Auction Implementation

At the beginning of the auction the auctioneer announces the item being auctioned and the closing time of the auction. Periodically, the auctioneer also announces the current time. Bidders submit bids to the auctioneer, who announces each bid. At the close of the auction, the auctioneer announces the winning bid and the winning bidder.

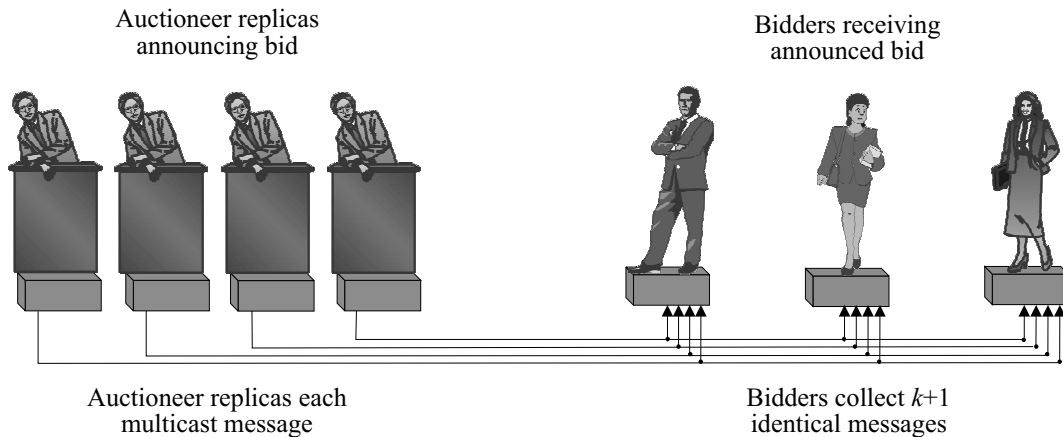


Figure 1. Replication of the auctioneer

The auctioneer object is replicated for fault tolerance, and all communication is performed using the SecureRing group communication system [10, 11], as described in Section 3.1. The use of the SecureRing protocols in addition to the replication mechanisms ensures that the auction service operates correctly despite collusion by some auctioneer replicas and bidders. The properties provided by SecureRing ensure that all bidders receive the same auction information, and that bidders cannot masquerade as other bidders or deny having submitted a bid.

Information regarding each auction is publicly available, such as the item being auctioned and the time allotted for the auction. The auctioneer replicas must reside on separate processors, but a processor may host both an auctioneer replica and a bidder object. A correct object always follows the protocol; a malicious object may behave in any arbitrary manner; thus, our fault model is that of *Byzantine* faults. We denote the total number of processors in the system by n and the total number of processors on which malicious objects reside by k , and require that $k \leq \lfloor (n-1)/3 \rfloor$.

The auction is implemented in Java, while the SecureRing protocols are implemented in C. The Java Native Interface is employed to provide a Java-based interface that maps to the underlying Unix socket-based interfaces of the SecureRing system.

3.1 SecureRing System

All communication in the auction application is performed by the SecureRing system [10, 11]. The SecureRing system provides reliable, total ordering of messages despite malicious (Byzantine) faults, provided that the number of Byzantine processors does not exceed $\lfloor (n-1)/3 \rfloor$, where n is the number of processors in the system.

The SecureRing system includes a message ordering protocol and a group membership protocol. Regular mes-

sages are generated by the application and are delivered in total order to the application. When a membership change occurs (*e.g.*, when a processor fails or a new processor joins), a membership change message is delivered to the application at the point in the message delivery order where the change occurred. Each membership change message initiates a new configuration.

The SecureRing message delivery protocol uses a combination of message digests and digital signatures to provide *secure delivery* for each configuration C , which is defined in terms of the following properties:

- **Integrity.** For any message m , every correct processor p delivers m at most once.
- **Authentication.** For any message m that contains the identifier of a correct processor p , a correct processor q delivers m only if m was originated by p .
- **Uniqueness of Messages.** If a correct processor p delivers a message m in configuration C , then no correct processor q delivers a mutant message m' in C having the same identifier as m but different contents.
- **Reliable Delivery.** If correct processors p and q are both members of configuration C , no configuration change occurs, and p originates a message m , then q delivers m . If correct processors p and q both install consecutive configurations C_1 and C_2 , and p originates a message m in C_1 , then q delivers m in C_1 .
- **Total Order of Messages.** If correct processors p and q both deliver messages m_1 and m_2 in configuration C , then p delivers m_1 before m_2 if and only if q delivers m_1 before m_2 .

The object group interface of the SecureRing system allows an application object to invoke the message delivery and group membership services provided by SecureRing.

An object can join or leave an object group, send a message to an object group, and receive a message that is addressed to its object group.

3.2 Replication

The auctioneer object is replicated, and the auctioneer replicas form an object group. The bidder objects collect messages from the auctioneer replicas. When a bidder has received the same message from $k + 1$ auctioneer replicas, the bidder accepts the message as valid. Replication of the auctioneer object is illustrated in Figure 1.

3.3 The Timekeeper Object

```

Upon startup:
state:= INTERIM
do forever
  update current time
  if state = IN_AUCTION
    send current_time message to auctioneer group

Received signal to start auction:
send start_auction message to auctioneer group
state:= IN_AUCTION

When auction ends:
state:= INTERIM

```

Figure 2. Pseudocode for the timekeeper object

The timekeeper object provides a source of the current time; it must be trusted to provide a reasonable approximation of the current time. The timekeeper object can be in one of two states: INTERIM or IN_AUCTION. It starts in the INTERIM state, and when it is near the time to begin an auction, it transitions to the IN_AUCTION state and sends a *start_auction* message to the auctioneer object group. The *start_auction* message contains the identifier of the auction and the start time for the auction. While in the IN_AUCTION state, the timekeeper object periodically sends a *current_time* message containing the current time to the auctioneer object group. At the close of an auction, the timekeeper shifts to the INTERIM state. Pseudocode for the timekeeper object is given in Figure 2.

3.4 The Auctioneer Object

An auctioneer object can be in one of three states: STARTUP, INTERIM, or IN_AUCTION. An auctioneer object starts in the STARTUP state and proceeds to join the auctioneer object group. When an auctioneer in the STARTUP state receives a *start_auction* message from the timekeeper, it records the auction identifier and the start

```

Upon startup:
state:= STARTUP
join auctioneer group

When receive message from timekeeper:
if (message type = start_auction and
state = STARTUP)
  save auction identifier and start time
  compute closing time and save
  state:= INTERIM
else if (message type = current_time)
  if (state = INTERIM)
    update local time
    if (local time ≥ auction start time)
      send open message to bidder group
      and auctioneer group
      send current_time message to bidder group
    else if (state = IN_AUCTION )
      send current_time message to bidder group
      and auctioneer group

When receive message from auctioneer replica:
switch (message type)
case current_time:
  if (received identical current_time messages from
 $k + 1$  auctioneer replicas and
state = IN_AUCTION)
    update local time
    if (local time ≥ closing time)
      send close message to bidder group
      and auctioneer group
case open:
  if (received identical open messages from  $k + 1$ 
auctioneer replicas and state = INTERIM)
    state:= IN_AUCTION
    set high bid and high bidder to NULL
case announce:
  if (received identical announce messages from
 $k + 1$  auctioneer replicas and
state = IN_AUCTION)
    update high bid and high bidder
case close:
  if (received identical close messages from  $k + 1$ 
auctioneer replicas and state = IN_AUCTION)
    send winner message to bidder group
    and auctioneer group
    state:= INTERIM

When receive submit message from bidder:
if (amount > high bid and state = IN_AUCTION)
  send announce message to bidder group
  and auctioneer group

```

Figure 3. Pseudocode for the auctioneer object

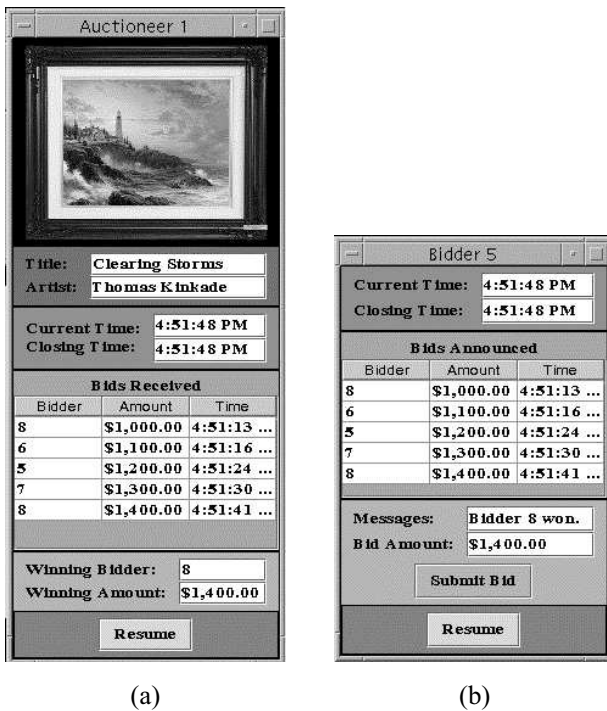


Figure 4. The graphical user interface to the auctioneer and to the bidder. Clearing Storms ©2000 Thomas Kinkade, Media Arts Group Inc., San Jose, CA, used by permission

time for the auction, computes and records the closing time for the auction, and shifts to the INTERIM state.

In the INTERIM state, when the auctioneer receives a *current_time* message from the timekeeper, it updates its local time value. When its local time becomes greater than or equal to the start time of the auction, the auctioneer object sends an *open* message to both the bidder group and the auctioneer group. The *open* message contains the auction identifier, the starting bid, and the closing time of the auction. The auctioneer then sends a *current_time* message to the bidder group. When the auctioneer receives identical *open* messages from $k + 1$ auctioneer replicas, the auctioneer shifts to the IN_AUCTION state and initializes its local values for the high bid and the high bidder to null.

When an auctioneer replica in the IN_AUCTION state receives a *current_time* message from the timekeeper, it sends a *current_time* message to both the bidder group and the auctioneer group. When an auctioneer replica receives identical *current_time* messages from $k + 1$ auctioneer replicas, it updates its local time value. During the auction, a bidder submits a bid by sending a *submit* message containing the bid amount. When an auctioneer replica in the IN_AUCTION state receives a *submit* message from a bidder, and the bid amount is greater than the current high bid, it sends an *announce* message to the bidder group and to the auctioneer group. The *announce* message contains

the identifier of the bidder, the bid amount, and the time that the bid was received. When an auctioneer replica receives identical *announce* messages from $k + 1$ auctioneer replicas, it updates its local high bid and high bidder values. When an auctioneer replica's local time value becomes greater than or equal to the auction closing time, it sends a *close* message to both the bidder group and the auctioneer group. When an auctioneer replica receives identical *close* messages from $k + 1$ auctioneer replicas, it announces the winner of the auction by sending, to the bidder group, a *winner* message that contains the bidder identifier and amount of the highest bid received, and shifts to the STARTUP state. Pseudocode for the auctioneer object is given in Figure 3.

The graphical user interface to the auctioneer object displays the item being auctioned and the bidding history, as shown in Figure 4(a). After the close of the auction, the winning bid and bidder are displayed.

3.5 The Bidder Object

```

Upon startup:
state:= INTERIM
join bidder group

When receive message from auctioneer replica:
switch (message type)
case open:
if (received identical open messages from k + 1
    auctioneer replicas and state = INTERIM)
state:= IN_AUCTION
case current_time:
if (received identical current_time messages from
    k + 1 auctioneer replicas)
update local time
case announce:
if (received identical announce messages from
    k + 1 auctioneer replicas)
record bid
case close:
if (received identical close messages from
    k + 1 auctioneer replicas)
state:= INTERIM
case winner:
record winning bid and bidder

When receive signal to submit bid:
send submit message to auctioneer group

```

Figure 5. Pseudocode for the bidder object

The bidder object can be in one of two states: INTERIM and IN_AUCTION. A bidder object starts in the INTERIM state and joins the bidder group. When a bidder receives identical *open* messages from $k + 1$ auction-

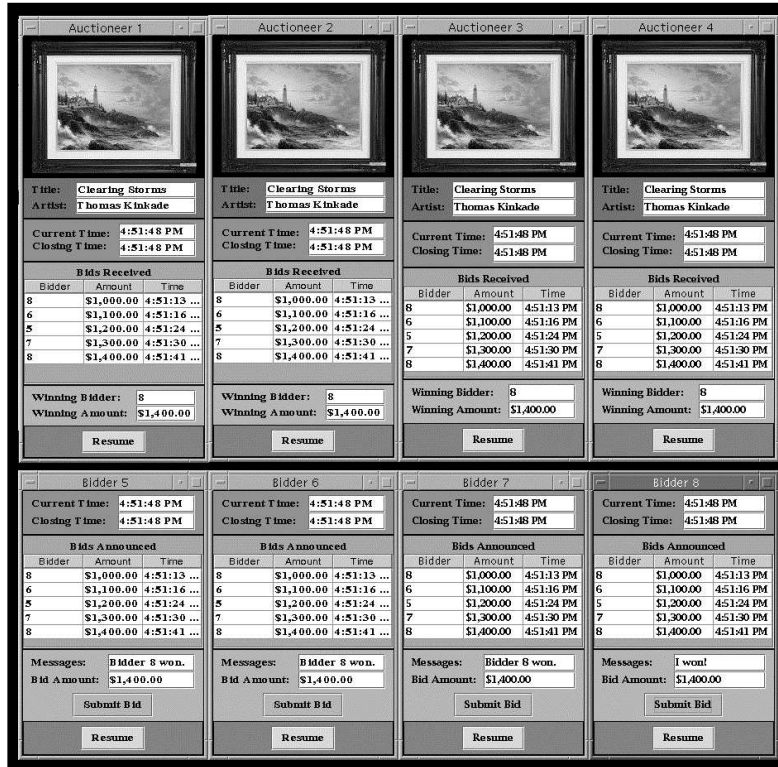


Figure 6. Multiple auctioneer replicas and bidders at the end of an auction. Clearing Storms ©2000 Thomas Kinkadee, Media Arts Group Inc., San Jose, CA, used by permission

eer objects, the bidder transitions to the `IN_AUCTION` state. While in the `IN_AUCTION` state, a bidder collects *current time* and *announce* messages that are sent by the auctioneer replicas and, once identical messages from $k + 1$ auctioneer replicas are received, the bidder considers the message as valid. The bidder may submit bids while in the `IN_AUCTION` state by sending a *submit* message containing the bid amount to the auctioneer group. When the bidder receives *close* messages from $k + 1$ auctioneer replicas, the bidder shifts to the `INTERIM` state. When the bidder receives *winner* messages from $k + 1$ auctioneer replicas, the bidder records the winning bidder and the amount of the winning bid. Pseudocode for the bidder object is given in Figure 5.

The graphical user interface to a bidder object displays the bidding history and allows the bidder to submit a bid, as shown in Figure 4(b). After the close of the auction, the winning bid and bidder are displayed. A complete view of multiple auctioneer replicas and bidders at the end of an auction is shown in Figure 6.

4. Auction Security

The auction operates correctly despite malicious behavior and/or collusion by some bidders and auctioneer replicas.

The properties of the SecureRing protocols ensure that all bidders receive the same auction information, all bids submitted within the auction period are included, and no bids submitted outside the period are included. Further, bidders cannot masquerade as other bidders and bidders cannot deny having submitted a bid; these guarantees are ensured by the use of message digests and digital signatures in the SecureRing protocols.

We now describe how the auction application achieves the properties of a secure English auction as described in Section 2., provided that $k \leq \lfloor (n - 1)/3 \rfloor$.

- *Inclusion: all valid bids submitted during the auction period are included.*
Exclusion: no bids submitted outside the auction period are included.

A correct auctioneer accepts bids only in the `IN_AUCTION` state. A correct auctioneer shifts in (out) of the `IN_AUCTION` state only if it has received identical *open* (*close*) messages from at least $k + 1$ auctioneers, which must include at least one correct auctioneer. A correct auctioneer updates its local high bid and high bidder values only if it has received identical announce messages from at least $k + 1$ auctioneers. All messages are delivered reliably and in total order due to the properties of the SecureRing proto-

cols, and so all correct auctioneers see the same sequence of bids during the auction period, which are exactly the bids submitted during the auction period.

- *Uniform receipt: all bidders receive the same auction information.*

A bidder accepts as valid identical messages from at least $k + 1$ auctioneer replicas, which must include at least one correct auctioneer. All messages are delivered reliably and in total order due to the properties of the SecureRing protocols, and so all bidders see the same sequence of messages during the auction period.

- *Ordering: bids are processed in the order in which they are submitted by the bidders.*

All messages are delivered reliably and in total order due to the properties of the SecureRing protocols. Therefore, all correct auctioneers will announce the bids in the order in which they were sent by the bidders.

- *Authentication: bidders cannot masquerade as other bidders.*

Non-repudiation: bidders cannot deny having submitted a bid.

If a bidder sends a *submit* message, then all correct auctioneers will receive the *submit* message. By the authentication property of SecureRing, the *submit* message is guaranteed to have been sent by the bidder, and the bidder cannot deny having sent the message.

5. Conclusion

We have defined the properties of a secure English (open bid, ascending price) auction, and have described a secure auction application. We have shown how our auction implementation achieves the properties of a secure English auction, provided that the number k of processors hosting malicious bidders or auctioneer replicas satisfies $k \leq \lfloor (n-1)/3 \rfloor$, where n is the total number of processors in the system.

Acknowledgments

The authors would like to thank Priya Narasimhan and Mike Reiter for insightful discussions, and the anonymous referees for their helpful comments.

References

- [1] M. K. Franklin and M. K. Reiter. The design and implementation of a secure auction service. *IEEE Transactions on Software Engineering*, 22(5), 1996, 302–312.
- [2] M. Harkavy, J. D. Tygar, and H. Kikuchi. Electronic auctions with private bids. In *Proceedings of the 3rd USENIX Workshop on Electronic Commerce*, Boston, USA, 1998, 61–73.
- [3] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, Denver, USA, 1999, 129–139.
- [4] S. Subramanian and M. Singhal. Real-time aware protocols for general e-commerce and electronic auction transactions. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, Austin, USA, 1999, 65–70.
- [5] M. Jakobsson and A. Juels. Mix and match: Secure function evaluation via ciphertexts. In *Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security, Lecture Notes in Computer Science 1976*, Kyoto, Japan, 2000, 162–177.
- [6] K. Viswanathan, C. Boyd, and E. Dawson. A three phased schema for sealed bid auction system design. In *Proceedings of the 5th Australasian Conference for Information Security and Privacy, Lecture Notes in Computer Science 1841*, Brisbane, Australia, 2000, 412–426.
- [7] S. G. Stubblebine and P. F. Syverson. Fair on-line auctions without special trusted parties. In *Proceedings of the Third International Conference on Financial Cryptography, Lecture Notes in Computer Science 1648* Anguilla, British West Indies, 1999, 230–240.
- [8] Y. Atzmony and D. Peleg. Distributed algorithms for English auctions. In *Proceedings of the 14th International Workshop on Distributed Computing, Lecture Notes in Computer Science 1914*, Toledo, Spain, 2000, 74–88.
- [9] E. Magkos, M. Burmester, and V. Chrissikopoulos. An equitably fair on-line auction scheme. In *Proceedings of the 1st International Conference on Electronic Commerce and Web Technologies, Lecture Notes in Computer Science 1875*, London, UK, 2000, 72–83.
- [10] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. The SecureRing protocols for securing group communication. In *Proceedings of the 31st IEEE Annual Hawaii International Conference on System Sciences, Volume 3*, Kona, USA, 1998, 317–326.
- [11] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. The SecureRing group communication system. *ACM Transactions on Information and System Security*, 4(4), 2001, to appear.