Secure Auctions in a Publish/Subscribe System *

Dawn Xiaodong Song
Carnegie Mellon University
skyxd@cs.cmu.edu

Jonathan K. Millen
SRI International
millen@csl.sri.com

Abstract

We present an approach to provide a fault-tolerant and secure service for sealed-bid auctions. The solution is designed for a loosely coupled publish/subscribe system. It employs multiple auction servers and achieves validity and security properties through application of secret-sharing methods and public-key encryption and signatures. It can tolerate Byzantine failures of one third of the auction servers and any number of bidders. A verification of the desired properties has been machine-checked using PVS. This work also provides insight and useful experience in techniques for specifying and verifying this type of system.

1 Introduction

The transition from traditional financial procedures to novel electronic and digital procedures is taking place worldwide at a surprisingly high speed. Electronic commerce systems, such as electronic trading, electronic banking, and electronic exchanges are becoming critical systems for society. As is the case with the traditional forms of critical systems, electronic commerce systems often require safety and reliability guarantees. They must be scalable and adaptable. They also require security properties such as secrecy, anonymity, and non-repudiation.

It's also commonly agreed that formal specification and verification are needed to provide solutions of this kind [15] [8] [10]. Many hand-checked protocols are found to be flawed via formal methods after they are proposed[5] [9] [6]. But there is still a lack of instructive experience and a systematic way of combining system building blocks and formal specification and verification techniques to provide a real solution.

Motivated by these problems, we studied one of these electronic commerce systems, sealed-bid secure auction service. A sealed-bid auction is one in which secret bids are issued for a certain item, and when the bidding is closed, the bids will be opened and the winner will be chosen according to certain publicly known rules.

*This work was supported by the U.S. Government under contract no. F30602-96-C-0291

Sealed-bid auctions are used in auctioning of various contracts, and in the sale of different types of goods, such as artwork and real estate [4][14].

Besides efficiency and scalability, sealed-bid auctions have strong security requirements. The identity of the bidders and the contents of the bids should not be revealed until the bidding is closed. After the bidding is closed, no more bids should be accepted as valid bids. The auction service should be able to tolerate a certain degree of corruption of the insiders in the auction house and the maliciousness of some bidders. In an internet environment, it is necessary to provide the required functional and security properties in the face of unreliable network communication and random failures of important components such as auction servers.

Franklin and Reiter have given a solution in the context of monetary bids [4]. Their solution is focused on using a cryptographic technique to provide protections to monetary bids, such as digital cash bids. It inherits certain properties from the digital cash scheme used for the bids. In their solution, every bidding message and auction server synchronization message requires atomic multicast [13] primitives, which can be a bottleneck in a large system.

In this paper we present a new approach which is built on a loosely coupled architecture and does not require atomic multicast. Loosely coupled publish/subscribe architectures have been widely used for scalable, adaptable distributed systems [11]. Their flexibility makes them a desirable infrastructure for many applications, but they generally lack fault tolerance and security support in malicious environments. Our challenge is to integrate fault tolerance and security in a loosely coupled publish/subscribe architecture in a systematic way and use formal specification and verification to increase the assurance of the design correctness[17].

Our solution is based on the the direct application of secret sharing and public key encryption. It can tolerate Byzantine failures of one third of the auction servers and any number of bidders. It provides a bid receipt service, which is often desirable in financial activities, and can be used by the bidder to prove that a bid was entered before the bidding was closed. We use PVS for formal specification and verification of the system and the properties [12]. A resulting prototype is in process to demonstrate the efficiency and scalability of the system.

The rest of the paper is organized as follows: the desired properties of the auction are summarized in the next section. In Section 3, we present the basic building blocks of the system and the cryptographic primitives needed in the design. In Section 4, we give an informal description of the protocol in detail. In Section 5, we give an overview of the formal specification of the system and some abstraction techniques. In Section 6, we list the desired system properties as specified in PVS and explain how we used PVS to prove these properties. Some issues are discussed in Section 7.

2 Auction Properties

The auction scheme is designed for any number of bidders and auction servers (also called auctioneers). Some of the auction servers and bidders may be faulty by either intentionally or incompetently failing to follow the specification of the protocol. The failure model and other environmental assumptions are discussed in detail later.

The desired properties of the auction are as follows:

1. The bidding period starts only if at least one good auction server decides that it should.
2. A good auction server stops accepting bids only after at least one other good auction server decides that the bidding period should be closed.
3. The identity of the bidders and the content of their bids are not revealed until the bidding is closed.
4. After the bidding period is closed, no more bids are accepted as valid.
5. Bidders are provided with evidence to prove that their bids are accepted before the bidding is closed.
6. Winning bid will be determined according to certain publicly known rules.

At the end of the auction, a winning bid is selected. Guarantees regarding the authenticity, nonrepudiation, and collectability of the bids are not provided by the protocol itself, but those issues can be addressed separately through construction of the bid contents.

3 Building Blocks

The three architectural components of the system are:

- a loosely coupled publish/subscribe system,
- a set of cryptographic primitives, and
- an auction protocol.

The first two of these are summarized below. The principal contribution of this paper is the design and verification of the auction protocol, as described in subsequent sections.

3.1 System Characteristics

3.1.1 Loosely Coupled Systems Loosely coupled systems have been developed to meet the need for large-scale survivable distributed systems [11]. The distinction between a loosely coupled system and a tightly coupled one lies in the way they handle process groups [1]. In a tightly coupled system there is a strong notion of group, sharing a common view of the group membership and the state of the system. A tightly coupled system often requires reliable multicast and atomic multicast [13].

The group membership protocol and reliable and atomic multicast primitives are complex and expensive to implement and can be a bottleneck of a system.

Loosely coupled systems, by contrast, do not need a strong notion of group membership. Instead of atomic multicast, they often use a publish/subscribe infrastructure where components acting in the role of publishers or subscribers communicate through a virtual bus (often called an “infobus”). Their great flexibility, adaptability and efficiency have made such systems suitable for very large and wide-area networks.

3.1.2 Publish/Subscribe Architecture In a publish/subscribe system, messages have a subject and a content field. Publishers publish messages under certain subjects. Subscribers subscribe to subjects of interest and receive the messages that are published under those subjects. Publish/subscribe systems are flexible because the subjects and contents of messages are minimally constrained by the core communication architecture. Subjects may have hierarchically organized, application-defined modifiers or subtopics, and the format of the message content can be defined freely according to the needs of the applications. Publish/subscribe also provides anonymity of publishers and subscribers.

For the auction scheme, there will be an auction subject, with modifiers identifying a particular auction and indicating whether the message is intended for auction servers or bidders. Auction Servers and bidders both publish and subscribe to appropriate message subjects as defined by the protocol. For each particular auction, there is a fixed set of auction servers of known size.

The subject field and subscription mechanism cannot be depended upon to support security objectives such as authenticating authorized publishers or restricting distribution of particular types of messages. For these and other security functions, we make use of additional cryptographic services.

3.1.3 Failure Model The failure model has two aspects: the reliability of message delivery in the network and the correctness of infobus clients, either auction servers or bidders.

The network is not assumed to be totally reliable. Messages can be delayed or lost or received out of order. However, the protocol is not designed for arbitrary network failure or indefinite denial of message delivery. It would not make sense to assume that an attacker can intercept any and all messages, since then the attacker can simply intercept all bidding messages from other bidders and only let its own bid go through.

It is assumed that published messages will be delivered to a sufficiently large portion of the network within a bounded time. That is, any routing failures or denial of service attacks, whether they are permanent or intermittent, can affect only relatively small segments of the network. By “relatively small,” we refer to the proportion of auction servers that may be affected. Since nothing is said about order of delivery, this assumption does not fall precisely into previously defined categories of "unreliable" or "reliable" communication in sources such as [4] and [11].

The second aspect of the failure model is the possible dishonesty of auction servers, possibly in collusion with bidders. We adopt a Byzantine failure model in which faulty auction servers may depart from the auction server protocol, withhold messages expected from it, subscribe to all auction-related messages, and publish all kinds of auction-related messages. A bidder may also be faulty and misbehave by submitting improper bids or publishing them at improper times.

A "good" auction server is one that is not faulty *and* lies in a segment of the network where messages published to other good auction servers will be received by all good auction servers in a bounded time. In practice, it may be necessary to send messages repeatedly to ensure delivery, and this can be a normal function of the basic publish/subscribe transmission protocol. The bounded-time assumption is discussed further in the Issues section at the end.

We assume that at most a specified number t of the n auction servers are not good, and that $n \geq 3t+1$. Any number of bidders may be faulty or isolated in parts of the network behind unreliable routers. Some bids may be lost for this reason.

3.2 Security Support

3.2.1 Public Key Infrastructure The protocol will make use of a public-key cryptosystem that must be used by auction servers and bidders for encrypting and signing messages, as called for in the protocol. We assume that there is a certification authority that can provide public key certificates prior to the auction. Implementing a practical public-key certification infrastructure is nontrivial, but this task is separable from the conduct of the auction. In fact, there may be many services other than an auction service that would make use of common key management facilities.

One auction-service-specific function is required of the certificate authority: the certificate for an auction server's public key should indicate that its role as an auction server is authorized.

3.2.2 Secret Sharing We also need a threshold secret sharing scheme. An (m, n)-threshold scheme permits a message to be projected onto n *shares* such that any m of them can be combined to reconstruct the original message, but less than m of them cannot. Several algorithms for this are given in Section 23.2 of [19].

4 Protocol Description

We assume that there are a set of n auction servers, denoted by S_1,...,S_n. The number n is fixed for a given auction. We assume that $n \geq 3t + 1$. For brevity, we refer to auction servers as "servers," though technically they are "clients" on the infobus. S_i has server ID s_i. There may be any number of bidders B_j, with identifiers b_j. The auction has a unique auction ID, denoted as *aid*.

All messages relating to this auction are published under an "auction" subject qualified by the auction ID. Some messages are intended solely for auction servers or bidders, and for efficiency that fact may be indicated as a subject modification as well. From an abstract or security point of view, it does not matter whether a field is part of the subject or part of the content of a message, and we assume that hostile parties can eavesdrop on all messages.

For simplicity of the representation, we introduce some shorthand denotations.

For any message a, $[a]^i$ is the encryption of a by server S_i's public key. It is assumed that any auction participant can look up and use S_i's public key given s_i.

For any message a, $[a]_i$ is a signed by server S_i's private key. We assume that a is recoverable from $[a]_i$, and that the signature can be checked by any participant given s_i.

All server messages in the protocol are signed, so that other servers will know they are authentic. This is important to determine subsequent server actions and to justify inferences about the state of good servers. Authentication of bids is not indicated in the protocol because it affects only the internal structure of bids, and it matters only for bid evaluation, which occurs after the protocol as specified has concluded.

We use a $(t+1, n)$-threshold sharing scheme, where t is the maximum tolerable number of faulty servers. $\mathsf{SSF}_i(s)$ is the ith share of a secret s.

A server's state transitions are depicted in Figure 1. A bidder's state transitions are depicted in Figure 2.

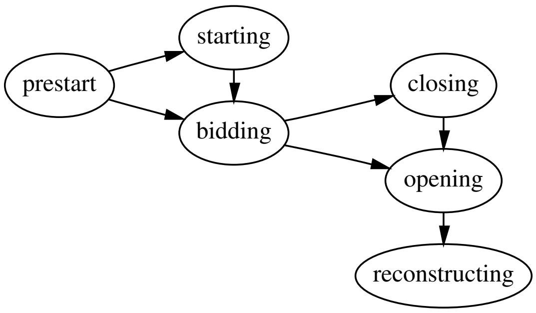

Figure 1: Server State Transitions

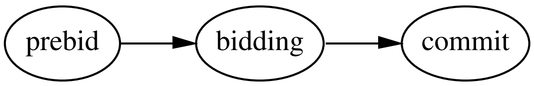

Figure 2: Bidder State Transitions

S.1 Starting the bidding When server S_i decides that the bidding should be started, it publishes a *start* message: *aid*, s_i, $[\mathit{aid}, \mathsf{start}]_i$. When S_i has received start messages from at least $t+1$ different other servers, it considers the bidding started and starts to accept bidding messages from bidders.

B.1 Submitting bids Suppose a bidder B_j decides to submit a bid y_j. The format of y_j will be discussed later.

B_j breaks y_j into shares $x_{ij} = \mathsf{SSF}_i(y_j)$, for $i = 1, ..., n$. Then B_j generates the *bid* message:

$$M_j = \mathit{aid}, b_j, [x_{1j}]^1, ..., [x_{nj}]^n.$$

This message is published to all servers.

S.2 During bidding When server S_i receives a bid M_j from a bidder B_j during the bidding, it publishes a *receipt*:

$$\mathit{aid}, b_j, s_i, [\mathsf{hash}(\mathit{aid}, M_j)]_i$$

where hash may be any standard one-way hashing function.

B.2 Committing the bids When B_j receives a receipt from S_i, it checks the validity of the receipt by checking the signature on the hash value. After B_j receives valid receipts from at least $2t + 1$ different servers, it enters its *commit* phase. Until then, it will either wait or periodically retry submitting the bid. We assume, essentially as part of the definition of a "good" server, that all good servers will eventually receive and acknowledge a correctly formatted bid.

S.3 Closing the bidding When S_i decides that the bidding should be closed, it publishes a signed *close* message:

$$\mathit{aid}, s_i, [\mathit{aid}, \mathsf{close}]_i.$$

When S_i has received close messages from at least $t + 1$ different other servers, it considers the bidding closed and stops accepting any more bidding messages from the bidders.

Suppose S_i received L_i bids in total. Let R_i be the set of indices of the bidders whose bids were received by S_i. Thus, R_i is of size L_i. For each $k \in R_i$, S_i decrypts its share of B_k's bid, namely x_{ik}.

It then publishes a *fingerprint* of the set of bids that it has received:

$$\mathit{aid}, s_i, [\mathsf{hash}(\mathit{aid}, \{(b_k, x_{ik}, M_k)\}_{k \in R_i})]_i$$

.

The fingerprint contains a signed hash of a list of triples, one for each received bid; each triple has the bidder ID, S_i's bid share, and the complete bid message. (Faulty servers may or may not send out a fingerprint message, but if they do, it is received by all good servers.)

S.4 Opening the bids After a bounded time, all the good servers should have stopped receiving bids, and have published their fingerprints. Since there are at most t faulty servers, there are at least $n - t$ fingerprints published.

After a bounded additional time, each good server S_i will have received fingerprint messages from all other good servers. They republish all the fingerprint messages that they have received. The inconsistent messages will be considered as from faulty servers and will be discarded. So all good servers will have the same set of fingerprint messages. Then it publishes its *bid-set* message, containing the information that was hashed to compute the fingerprint. The bid-set message is:

$$aid, s_i, [\{(b_k, x_{ik}, M_k)\}_{k \in R_i}]_i.$$

S.5 Reconstructing the bids After another bounded additional interval, each good server S_i will have received all bid-set messages sent by all other good servers.

When S_i receives a bid-set message from S_j, it first checks whether it matches the fingerprint from S_j by computing the hash value. If they don't match, it means S_j is faulty and that bid-set message is discarded by S_i (and all other) good servers.They republish all the bid-set messages that they have received. The inconsistent messages will be considered as from faulty servers and will be discarded. So all good servers will have the same set of bid-set messages.

S_i reconstructs the bid from B_j as follows. Let T_{ik} be the set of indices of servers S_j from whom a bid-set message with M_k has been received by S_i.

For each index $k \in T_{ik}$, S_i can extract S_j's share of B_k's bid y_k, namely x_{jk}, from the jth bid-set message, compute $[x_{jk}]^j$, and compare this with the value from the bid message M_k. If they match, the share x_{jk} is valid and can be used to reconstruct the bid y_k.

If T_{ik} contains at least $t + 1$ elements, then S_i combines those $t + 1$ shares to construct a value y'_k that should be equal to the bid y_k.

If there exists any j such that $[\mathsf{SSF}_j(y'_k)]^j \neq [x_{jk}]^j$, where $[x_{jk}]^j$ is taken from the bid message M_k, then S_i discards the bid from B_k.

In this way, S_i reconstructs a set of bids and selects a winner according to the publicly-known rule for the auction.

All the good servers will reconstruct exactly the same set of bids, because each of them received the same set of bid-set messages. The majority of the servers will agree on a selection, since good servers are in the majority, and that selection is declared the winner of the auction. Issues such as authentication and enforcement of the bids will be discussed in a later section.

5 Formal Specification of the System

The secure auction service system is a distributed system composed of asynchronous processes, namely, the auction servers and bidders. Systems and most programming language structures can be modeled as state machines[18]. A state machine consists of some encoding of the system state, and the next-state transition relationship.

Compositional reasoning and verification are often necessary and desired to simplify the complexity of a verification[3]. The state of a distributed system can be viewed as the composition of the local states of its component processes. The state transition relation, as well, can be decomposed into local state transitions per component.

Abstraction of the system structure, including communication and cryptographic primitives, is necessary for protocol level specification and verification. In this section, we describe how we use composition and abstraction techniques for the system specification.

For the secure auction service system, the global state is the composition of the local states of the components representing auction servers and bidders. Each of these components operates asynchronously according to a local state transition relation. There are two local transition relations, one for auction servers and one for bidders.

All auction servers have the same state structure, and so do all of the bidders. These structures are described in the next subsection.

The infobus is modeled using local state variables that record the sets of messages that have been published by each participant. The state of the infobus is the union of all of these locally-defined sets.

The global state structure is summarized schematically in Figure 3. The figure shows how the auction server state and the bidder state are decomposed into state variables. The infobus state also has components, each of which is derived as the union of corresponding local state components.

5.1 Abstraction of the Auction Server

An auction server is a local state machine with the state variables shown in Figure 3. The phase variable has one of the values *prestart, starting, bidding, closing, opening, reconstructing*. *wantStart* and *wantClose* are boolean variables that indicates when the auction server decides that it's time to start or close, respectively.

start_buffer, close_buffer, bids_buffer, fingerprint_buffer and *bidset_buffer* are sets of IDs identifying servers and bidders from whom messages of these kinds have been received.

openBid is the set of IDs identifying bidders whose bid shares have been opened by this auction server, i.e., those that are included in its bid-set message.

holdShare_buffer is the set of all the shares that the auction server can decrypt from its *bids_buffer*. *holdBid_buffer* is the set of all the bids that the auction server reconstructs at the end.

sendStart and *sendClose* are boolean variables that indicate when the auction server has already sent out start or close messages. *receipt_buffer* is the set of all the IDs of bidders whose bid it has acknowledged by a receipt.

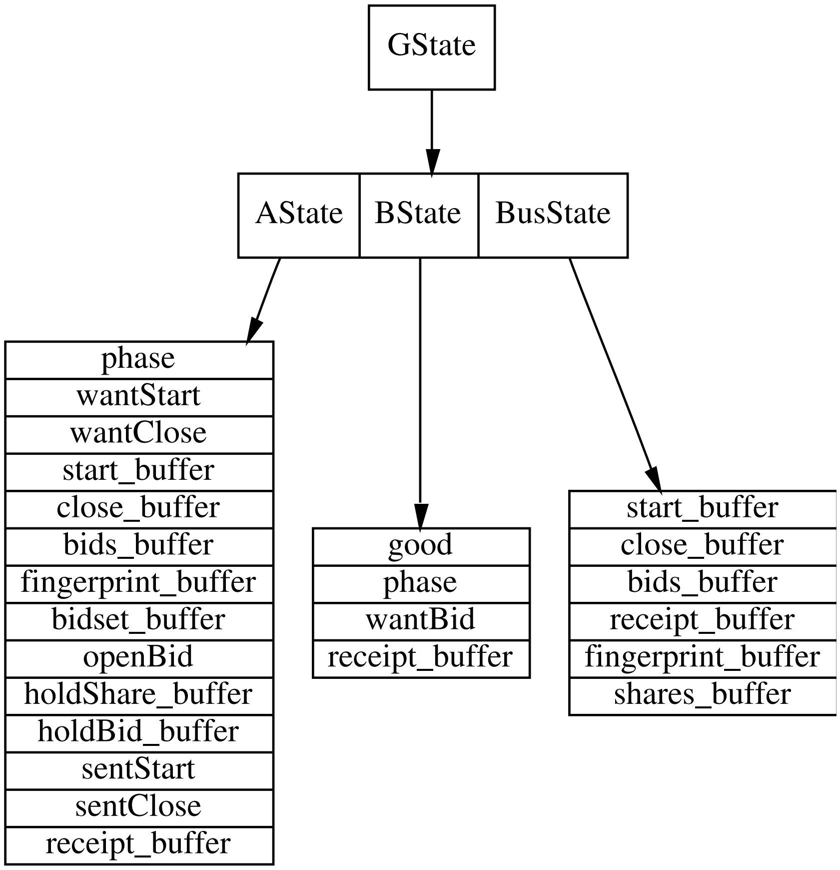

Figure 3: Global State Structure

5.2 Abstraction of the Bidder

A bidder is a local state machine with the local state variables shown in Figure 3. *good* is a boolean flag that indicates whether the bidder is "good," that is, if it follows the protocol specification. The *phase* variable has one of the values *prebid, bidding, commit*. *wantBid* is a boolean variable that indicates that when the bidder decides that it's time to submit its bid. *receipt_buffer* is the set of IDs of servers from whom the bidder has received a receipt.

5.3 Abstraction of the Publish/Subscribe Communication

The bus has a state with six components. Each component is a set of IDs of servers or bidders who have published messages of each type: *start_buffer, close_buffer, bids_buffer, receipt_buffer, fingerprint_buffer and shares_buffer*. These sets are computed from corresponding state variables in the local states of the servers and bidders.

In any state transition in which a message is published, that fact is recorded in the local state of the publisher, and appears also by definition in the state of the bus. A message can be received (as indicated in a local state variable) only if the message has previously been published, as recorded in the current bus state. This is a fact about the construction of the next-state transition relation. Also, by construction, each buffer set is nondecreasing.

While some state variables contain sets of messages, such messages are formalized as elements of a primitive type, so that the actual contents and formats of protocol messages are not explicitly represented in the specification. Instead, their essential properties are axiomatized.

6 Formal Specification and Verification of Security Properties in PVS

This section describes how the auction protocol was specified and verified using the PVS environment.

6.1 PVS Overview

PVS is a integrated environment for specification and automated verification developed at SRI [12]. PVS specification language is based on higher-order logic with a richly expressive type system. It supports standard theories of integers, sets, functions, and relations, as well as the ability to construct new abstract data types. The PVS theorem prover consists of a powerful collection of inference steps augmented with a library of decision procedures and the ability to add user-defined proof strategies.

A PVS specification is divided into *theories*, each defining a related set of data types and stating axioms and theorems about them. Data type declarations resemble those in a strongly-typed programming language. The bulk of the auction service specification is in a single theory that introduces types for the state data structures summarized above.

The subsections below show how the essential property of the shared secret function is axiomatized and how the auction service properties are stated. A few remarks about PVS notation should be sufficient to read these formulas.

The new data types include *ID, GID, BID*, and *trace*. The ID type consists of all auction server IDs, with a subtype GID of good server IDs. The BID type is for bidder IDs. A trace is, by definition, a sequence of global states beginning with an initial state and such that each consecutive pair of states is consistent with the transition relation.

Components of a structure are accessed by using the component names as functions. Local states are obtained from a global state by indexing on the ID, so that, for example, the holdBid component of server i in the global state g is `holdBid(astate(g)(i))`.

In PVS, a set can be represented by a boolean function. Thus, the formula $x \in \{y|G(y)\}$ would appear in the specification as `G(x)`, and the set itself is written `(G)`.

The shared-secret function invocation $\mathsf{SSF}_i(j)$ is written `SSF(i)(j)`, and `card` is the cardinality function.

6.2 Axiomatization of the Shared Secret Function

The mathematical properties of the threshold sharing scheme are captured by the following axiom, stating that at least $t+1$ shares of a bid must be held by a server S_i, as indicated in its *holdShare* state variable, in order for that server to hold the reconstructed bid, as indicated in its *holdBid* state variable. This is stated as true for every global state in a trace. The contents of *holdBid* are not affected or constrained by any other part of the specification.

```
holdBid_true: AXIOM
   FORALL (trace1:trace,j:nat,i:ID,b:BID):
holdBid(astate(trace1(j))(i))(myBids(b)) AND
good(bstate(trace1(j))(b)) <=>
(EXISTS (y:finite_set[below[N]]):
(FORALL(a:(y)):
holdShare(astate(trace1(j))(i))(SSF(myBids(b))(a))) AND
card(y)>t)
```

6.3 Invariants

The desired properties of the system are invariants; they are true of every reachable state, i.e., every state in a trace. They are proved inductively by showing that they are true in an initial state and preserved by all state transitions.

- ```
  Safe1: THEOREM
    FORALL (trace1: trace, j:nat, gid:GID):
  phase(astate(trace1(j))(gid))=bidding =>
  EXISTS (i:GID): (wantStart(astate(trace1(j))(i))
  ```

  The bidding period starts only after a good auction server decides that it should start.

- ```
  Safe2: THEOREM
    FORALL (trace1: trace, j:nat, gid:GID):
  phase(astate(trace1(j))(gid))=opening =>
  EXISTS (i:GID): (wantClose(astate(trace1(j))(i)))
  ```

  A good auction server stops accepting bids only after some good auction server decides that the bidding period should be closed.

- ```
   pss: THEOREM
  FORALL (trace1:trace,j:nat,i:ID,b:BID):
  holdBid(astate(trace1(j))(i))(myBids(b)) AND
  good(bstate(trace1(j))(b))  =>
  CLOSE_bid(trace1(j))
  ```

  Before the bidding is closed, the identity of the bidder and the bids of the bidder are not revealed. CLOSE_bid(g) is defined as true if all good servers in global state g have reached at least the opening phase.

- ```
   Uniform: THEOREM
  FORALL (trace1:trace,j:nat,i1:GID,i2:GID,b:BID):
  (holdBid(astate(trace1(j))(i1))(myBids(b)) AND i1/=i2
  AND Open_bid(trace1(j))
  AND good(bstate(trace1(j))(b))) =>
  holdBid(astate(trace1(j))(i2))(myBids(b))
  ```

  After the bids are reconstructed, all the good servers reconstruct the same set of bids.

- ```
   Close1: THEOREM
  FORALL (trace1:trace,j:nat,i:GID,b:BID):
  (holdBid(astate(trace1(j))(i))(myBids(b)) AND
  good(bstate(trace1(j))(b))) =>
  validBid(trace1(j))(b)
  ```

  After the bidding period is closed, no more bids can be accepted as valid bids. validBid(g)(b) is defined as true in a global state g if the bid from b has been accepted by at least one good server.

- ```
   commit: THEOREM
    FORALL (trace1:trace,i:GID,b:BID,j:nat):
  phase(bstate(trace1(j))(b))=commit
  ```

```
AND good(bstate(trace1(j))(b))
AND Open_bid(trace1(j))
 => holdBid(astate(trace1(j))(i))
```

If a good bidder commits, its bid is guaranteed to be reconstructed and taken into final consideration as a in-time bid.

7 Design and Modeling Issues

This section discusses some issues regarding assumptions and design choices that were made in the present protocol design.

7.1 Other properties of the Auction

At the conclusion of the protocol as presented, all good servers have opened the same set of bids and agreed on a winner. The identity of the bidder supplying that bid is not guaranteed by the protocol. Any authentication or nonrepudiation if needed can be provided by some other cryptographic primitives and the format of the bids which is application-specific.

7.2 Delivery of electronic goods

If the object of the auction is in electronic form, such as software or a postscript file, our original approach can be extended to secure delivery as follows. Every bidder will include a public key in its bid. Then the goods can be transmitted confidentially to the winner by using the public key provided in the winner's bid. This public key need not be certified, because it is in the interests of the winner to provide the correct key, and the good servers will agree on its value.

We might also ask where the file to be awarded was held prior to delivery to the winner. Rather than trust any one server to hold it, it can be split using a $(t+1, n)$ secret-sharing scheme among all servers. Each server will publish its own share encrypted by the winner's public key so that the winning bidder will receive enough shares to reconstruct the item, and a collusion of faulty servers will not be able to reconstruct it.

7.3 Externally Triggered Transitions

Certain state transitions occur as a result of the passage of time, based on assumptions about the reliability of good servers and network message delivery. Good servers decide to start the bidding and close the bidding according to a predefined date/time schedule for the auction or some external event. They consult a local system clock or receive some other events to trigger those state changes. The triggering events may be out of synchronization, but the protocol compensates for this by forcing good servers to undergo the phase change when it has received signal messages from $t+1$ other servers. The number $t+1$ means that at least one good server has sent out its signal.

Event-triggered state changes are indicated with boolean state variables. In the specification, they are set nondeterministically.

7.4 Time Bounds

A good server opens bids only when it knows that all good servers have stopped accepting bids and published their fingerprints. This knowledge comes not from having received any particular number of close or fingerprint messages, but rather from the time bound on actions of good servers and delivery of their messages. The transition to opening bids is triggered in the specification by a predicate on the global state testing whether all good servers have published their fingerprint messages.

The assumption that good servers can send messages to one another within a known time bound is a strong but reasonable assumption. The protocol will fail if some global outage (internet worms, satellite failure, etc.) affects a large portion of the network for an excessive time. We are investigating whether we can weaken the delivery assumption by making use of failure detectors or by assuming instead partial synchrony, where a time bound exists but is not known [2]. Alternatively, it may be adequate to recognize, when a known time bound passes, that an insufficient number of good servers has responded, and declare the auction invalid without compromising the bids. In the present protocol, if too many servers go out of communication, it is a liveness rather than a safety or security problem, since the bids will remain secret.

8 Conclusions

The motivation for this work was to understand whether it is possible to integrate fault-tolerance and security into loosely coupled publish/subscribe systems and to combine the system building blocks with formal techniques to provide possible solutions for electronic commerce systems, particularly a secure auction service.

We have accomplished these goals, and gained assurance in the correctness of the design through the use of an established specification and verification facility. One of the beneficial consequences of the verification activity was a better understanding of what assumptions to make about message delivery, leading us to a different category of "reliable" transmission that is reasonable for a publish/subscribe system.

We are in the process of implementing a prototype system demonstrating the design, using the Java Infobus application program interface.

Acknowledgements

Thanks to John Rushby for helpful discussions and advice. Thanks to Sergey Berezin and others at SRI for help with PVS.

References

[1] K. P. Birman. The process group approach to reliable distributed computing. *Comm. ACM*, 1993.

[2] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 1988.

[3] E.Clarke, D.Long, and K.McMillan. Compositional model checking. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 353–362, 1989.

[4] M. K. Franklin and M. K. Reiter. The design and implementation of a secure auction service. In *IEEE Security and Privacy Symposium*, pages 2–14. IEEE Computer Society, 1995.

[5] G.Lowe. Breaking and fixing the needham-schroeder public-key protocol using FDR. In *Proceedings of TACAS, Lecture Notes in Computer Science*, volume 1055, 1996.

[6] Li Gong, R.Needham, and R.Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of 1990 IEEE Symposium on Research in Security and Privacy*, 1990.

[7] L.Lamport and M.Pease. The byzantine generals problem. *ACM TOPLAS*, 1982.

[8] L.Paulson. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop*, 1997.

[9] M.Burrows, M.Abadi, and R.Needham. A logic of authentication. In *Proceedings of the Royal Society*, volume 426 of *A*, pages 233–271, 1989.

[10] Catherine Meadows. The NRL protocol analyzer: an overview. *Journal of Logic Programming*, 1996.

[11] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The information bus - an architecture for extensible distributed systems. *ACM Operating Systems Review*, 27(5):58–68, 1993.

[12] S. Owre, J. M. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Trans. on Software Engineering*, 21(2):107–125, February 1995.

[13] Michael Reiter. Secure agreement protocols: Reliable and atomic group multicast in rampart. In *2nd ACM Conference on Computer and Communications Security*, November 1994.

[14] R.McAfee and J.McMillan. Auctions and bidding. *Journal of Economic Literature*, 1987.

[15] John Rushby. Formal methods and their role in the certification of critical systems. Technical Report SRI-CSL-95-1, Computer Science Laboratory, SRI International, Menlo Park, CA, March 1995.

[16] John Rushby. Systematic formal verification for fault-tolerant time-triggered algorithms. In Mario Dal Cin, Catherine Meadows, and William H. Sanders, editors, *Dependable Computing for Critical Applications—6*, volume 11 of *Dependable Computing and Fault Tolerant Systems*, pages 203–222, Garmisch-Partenkirchen, Germany, March 1997. IEEE Computer Society.

[17] John Rushby and Friedrich von Henke. Formal verification of algorithms for critical systems. *IEEE Transactions on Software Engineering*, 19(1):13–23, 1993.

[18] Fred Schneider. Implementing fault-tolerant services using state machine approach: a tutorial. *ACM Computing Serveys*, 1990.

[19] B. Schneier. *Applied Cryptography*. Wiley, 1996.