**SCC.311 Coursework Part B Specification** [Revision 3]
**Due:** Friday Week 6 via Moodle
**Total marks available:** 40

You are asked to implement a distributed auctioning system using Java RMI. This auctioning system should consist of a central auctioning server and a client program.

### Level 3: Auction Logic (20 marks)

Implement server logic to handle *creating* and *managing* listings, and *bidding* on listed items. This auctioning system should use the exact interface shown below:

```
public interface Auction extends Remote {
 public int register(String email) throws RemoteException;

 public AuctionItem getSpec(int itemID) throws RemoteException;

 public int newAuction(int userID, AuctionSaleItem item) throws RemoteException;

 public AuctionItem[] listItems() throws RemoteException;

 public AuctionResult closeAuction(int userID, int itemID) throws RemoteException;

 public boolean bid(int userID, int itemID, int price) throws RemoteException;
}
```

You must implement every function, including user creation, new auction creation, the ability to list the auction items and make bids, and the ability to close auctions. The `register()` function must return a unique user ID for the newly-created user, which can then be used as a `userID` parameter to other functions in the interface which expect a user ID. The `newAuction()` function must return a unique auction ID which can then be used as the `itemID` in any other functions which expect an auction item ID. The class definitions of `AuctionItem`, `AuctionSaleItem`, and `AuctionResult`, are given at the end of this document.

You should also write a client program which is able to create new auctions and close them, list currently-open auctions, and make bids on auction items.

The auctioning server should be able to deal with requests from multiple different client program instances, and maintain the state of ongoing auctions. Note that you should use in-memory Java data structures to store all auction data (using a persistent storage solution will not gain additional marks, and may make the following coursework stage more difficult).

Remember to apply principles of objective oriented programming, such as identity separation and encapsulation.

### Level 4: Authenticated Auctioning (+ 20 marks)

Modify your system to provide asymmetric cryptographic authentication. For this level your auction interface must have two additional methods *challenge* and *authenticate* as shown below, as well as parameter changes to some of your existing functions:

```
public interface Auction extends Remote {
 public int register(String email, PublicKey pkey) throws RemoteException;

 public ChallengeInfo challenge(int userID, String clientChallenge) throws RemoteException;
```

```
 public TokenInfo authenticate(int userID, byte signature[]) throws RemoteException;

 public AuctionItem getSpec(int userID, int itemID, String token) throws RemoteException;

 public int newAuction(int userID, AuctionSaleItem item, String token) throws RemoteException;

 public AuctionItem[] listItems(int userID, String token) throws RemoteException;

 public AuctionResult closeAuction(int userID, int itemID, String token) throws RemoteException;

 public boolean bid(int userID, int itemID, int price, String token) throws RemoteException;
}
```

---

You must use RSA keys, which must be 2048 bits long, for all authentication operations, and you must use the `SHA256withRSA` algorithm for creating and verifying digital signatures. You can assume that the server's public key has been shared in advance and, in a common folder called `keys` located in the root directory of your submission.

The `ChallengeInfo` and `TokenInfo` classes must be defined as follows:

---

```
public class ChallengeInfo implements java.io.Serializable {
 byte response[]; // server's response (signature) to client's challenge
 String serverChallenge; // server's challenge to the client
}

public class TokenInfo implements java.io.Serializable {
 String token; // a one-time use token issued by server
 long expiryTime; // expiration time as a Unix timestamp
}
```

---

The `challenge()` function on the server will use the server's private key to generate a signature on the clientChallenge string provided by the client. The `authenticate()` function accepts a signature of the server's challenge (in `ChallengeInfo`) and returns a `TokenInfo` instance if that signature is valid. Each `TokenInfo` object contains a one-time use token which has an expiry time. The `authenticate()` function should invalidate any previous tokens for the corresponding client.

The server expects a valid token with any client request; otherwise, the server does not execute the request and returns null, false, or some other appropriate value. A valid token is one that was generated by the server for a given client, has not been used by the client before, and has not expired. In your server code, you should set each token to expire within ten seconds. The server should use minimal state to validate tokens. As in Level 3, you should only use in-memory Java data structures to store any user-and auction-related state on the server.

**Coursework submission instructions**

Your coursework will be partly marked using an automated test system. For the test system to work properly, your submission must be contained in a zip file and have the following:

1. A shell script called server.sh, in the root directory of your submission, which performs any set of operations necessary to get your server running.
2. An RMI service advertised with the name "Auction".
3. An RMI interface which exactly matches the specification given in this document.
4. Your entire system must be up and running within 5 seconds (our test client is launched 5 seconds after your `server.sh` script).
5. If you attempt level 4, a directory called 'keys' which contains the server's public key, stored in a file called `server_public.key`.
6. If you attempt level 4, your signature / verify methods must use the algorithm "SHA256withRSA".

You can test if your submission is valid by uploading your code, in a .zip file, to the web page `http://scc-311-24-rock.lancs.ac.uk/partB`. This will check whether the above basic checks pass, but does not perform any logic testing or provide any indication of your mark. **If our automated testing system is unable to test your code, you would usually receive a mark of zero.**

**Mark Scheme**

**Level 3**

Selling client: create listings with reserve, close listings, announce winner — 6 marks

Buying client: browse, bid — 6 marks

Management of listing and bidding data on server — 4 marks

Support for multiple buyers and sellers simultaneously transacting with the server — 4 marks

**Level 4**

3-stage challenge-response protocol — up to 12 marks

Correct access control implementation on auction state machine — 8 marks

**Additional class definitions**

The following classes are used by the `Auction` interface, and must be defined exactly as follows:

```
public class AuctionItem implements java.io.Serializable {
 int itemID;
 String name;
 String description;
 int highestBid;
}
```

```
public class AuctionSaleItem implements java.io.Serializable {
 String name;
 String description;
 int reservePrice;
}
```

```
public class AuctionResult implements java.io.Serializable {
 String winningEmail;
 int winningPrice;
}
```