SCC.311 Coursework Part B Specification
Due: Friday 5pm Week 6 via Moodle
Total marks available: 38%

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 (14%)

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 Integer register(String  email) throws  RemoteException;

  public AuctionItem getSpec(int itemID) throws RemoteException;

  public Integer 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 method, including user creation (i.e., registration), new auction creation, the abilityto list the auction items and make bids, and the ability to close auctions. The class definitions ofAuctionItem, AuctionSaleItem, and AuctionResult, are given at the end of this document.

To test your code, 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 deal with requests from multiple client program instances,and maintain the state of ongoing auctions. Note that you should use in-memory Java data structuresto store all auction data (using a persistent storage solution will not gain additional marks, and will make the following coursework stage more difficult).

Remember to apply principles of objective-oriented programming, such as identity separation and encapsulation. We'll be marking how elegant, clean and efficient your code is, so consider yourdesign carefully.

Level 4: Authenticated Auctioning (24%)

Modify your system to provide asymmetric cryptographic authentication. For this level, theauction interface methods are slightly updated, and there are also two additional methods: challenge() and authenticate(). The updates are highlighted with red.

Note: Please simply copy the interface definition below and paste it to your Auction.java file.

```
public interface Auction extends Remote {

    public Integer register(String email, PublicKey pubKey) 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 Integer 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 should use RSA keys (2048 bits) for all authentication, and you can assume that the server's public key has been securely stored in a file named 'serverKey.pub' in a 'keys' folder located in the root directory of your submission. Please use the storePublicKey() method given below to store the server's public key. Once a client registers its username (i.e., email address and its public key) with the server, from that point onwards, it must perform a **3-way authentication** before **any** interaction with the auction server.

A 3-way authentication involves a challenge() method invocation followed by an authenticate() method invocation by the client. The challenge() method implementation should use the server's private key to generate a signature on the clientChallenge string supplied by the client. The challenge() method should then return the signature generated by the server as well as a (random) challenge for the client as part of a ChallengeInfo object (see below).

In the final step of the 3-way authentication, the client must generate a signature on the server's challenge (i.e., serverChallenge) with its private key and call the authenticate() method to send its signature to the server. The authenticate() method implementation at the server must verify the client's signature, and if the verification is successful, the server must return a TokenInfo object containing a *one-time use* token and its expiration time.

The server expects a valid token with any client request; otherwise, the server does not execute the request and returns null or false (whichever is appropriate). 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 a **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.

Notice that a token (of type String) is now an argument to the getSpec(), newAuction(), listItemts(), closeAuction(), and bid() methods in the updated Auction interface of Level 4. Also, the register() method takes as argument the public key of the user.

The ChallengeInfo and TokenInfo classes must be defined **exactly** as follows. The ChallengeInfo provides the server's response to the client's challenge and the client's challenge to the server.

```
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
}
```

Your system must be able to also mitigate the following access control violations:

1) tampering with bids (one buyer modifying or stopping another buyer's bid),
2) closing the bidding by a user who did not create the auction,
3) accepting a bid that is the same or lower than the current bid.

You must implement logic to ensure the right level of access for different users to protect against the above attacks.

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 anyset 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 in a file named ‘serverKey.pub’. The key must be stored in Base64 encoding – you can use the ‘storePublicKey()’ method given at the end of this document to correctly store a public key at a given file path.
6. If you attempt level 4, your signature/verification methods should use the algorithm “SHA256withRSA”.

Marking Scheme:

Level 3

Be able to create a user, create an item, get an item’s spec and get listings of open auctions— 4 marks
Be able to bid and close an auction, returning the correct winner — 4 marks
Support for multiple buyers and sellers simultaneously transacting with the server — 4 marks
Clean, elegant, and efficient code – 2 marks

Level 4

Correct implementation of 3-stage challenge-response protocol – 6 marks
Use of minimal client authentication state and use of appropriate data structures — 6 marks
Validation of one-time use tokens – 6 marks
No access control violations – 4 marks
Clean, elegant, and efficient code – 2 marks

Additional class definitions:

The following classes are used by the Auction interface, and must be defined **exactly** as follows (please do not add constructors or any additional methods!):

```
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;
}
```

```
import java.io.FileOutputStream;
import java.util.Base64;
…
// Method to write a public key to a file.
// Example use: storePublicKey(aPublicKey, '../keys/serverKey.pub')
public void storePublicKey(PublicKey publicKey, String filePath) throws Exception {
        // Convert the public key to a byte array
        byte[] publicKeyBytes = publicKey.getEncoded();

        // Encode the public key bytes as Base64
        String publicKeyBase64 = Base64.getEncoder().encodeToString(publicKeyBytes);

        // Write the Base64 encoded public key to a file
        try (FileOutputStream fos = new FileOutputStream(filePath)) {
            fos.write(publicKeyBase64.getBytes());
        }
    }
```