Task

Your knowledge in using the Java cryptographic API and related features, and your understanding of cryptographic concepts such as public key cryptography.

You will write a client-server message board application that supports encryption and authentication of messages.

Client-server architecture, public and private keys

The system consists of a client and a server Java program, and they must be named Client.java and Server.java respectively. They are started by running the commands

```
java Server port
java Client host port userid
```

- specifying the hostname and port number of the server, and the userid of the client.
- The server program is always running once started, and listens for incoming connections at the port specified. When a client is connected, the server handles the request, then waits for the next request (i.e., the server never terminates). For simplicity, you can assume that only one client will connect to the server at any one time.
- Each user has a unique userid, which is a simple string like alice, bob etc. Each user is associated with a pair of RSA public and private keys, with filenames that have .pub or .prv after the userid, respectively. Thus the key files are named alice.pub, bob.prv, etc. These keys are generated separately by a program RSAKeyGen.java More details are in the comment of that program.
- It is assumed that the server already has the public keys of all legitimate users, and each client program user already has their own private key as well as the public keys of anyone to whom they want to send secret messages. They obtained these keys via some offline method not described here, prior to the execution of the client and server programs. **The client and server programs never create any new keys**.
- All the key files are in the same folder where the client/server programs run from. They must not be read from other folders. Your programs must not require keys that they are not supposed to have.

The server program

- The server keeps a collection of all the *posts* sent by all legitimate users. A "post" consists of three pieces of information: the userid of the sender (a string), the message (a string) which may or may not be encrypted (see the next point), and a timestamp (the suggestion is to use the java.util.Date object).
- The system allows both unencrypted posts that are intended for everyone, and encrypted posts that can only be decrypted by the intended recipient. If a post is encrypted, the message part would have been encrypted with RSA and the appropriate key of the intended recipient, then converted to a Base64 string. The sender userid and timestamp parts are not encrypted.
- Since this encrypted-and-converted message is also a string, the server treats it the same way as an unecrypted message. Note that neither the server nor anyone other than the intended recipient knows how to decrypt it or even who this encrypted message is for.
- The server keeps all the posts, in the order they are received. Initially (when the server is just started), it has no posts. For simplicity, we assume there are no persistent storage of these posts (so when the server program quits, all posts are lost). The posts are otherwise never removed.
- Upon the connection of a new client, the server first sends all the posts it currently has to the client. Then, it reads some information (e.g. a boolean variable or a string) from the client indicating whether it wants to post a message. If the client does not want to post a message, then the connection ends. Otherwise, the server receives the post (the sender userid, the possibly encrypted message, and the timestamp). It also receives from the client a signature, that is computed based on all three fields of the post and signed by the client (with the appropriate key) to prove their identity.
- After receiving the post and the signature, the server verifies the signature with the appropriate key. If the signature checks out, it accepts the post and adds it to its collection of posts. (The signature itself is not part of the post and is not stored.) If the signature does not verify, the post is discarded. In either case, the server should print the contents of the post (all three fields) and the accept/reject decision to the screen, just for debug purposes.
- The connection then ends and the server should wait for the next client. The server should not quit or terminate (even if the signature check fails).

The client program

- When the client program starts, it connects to the server to retrieve all the posts. For each post, it displays the sender userid and the timestamp, and handles the possibly encrypted message as follows. Since it does not know whether each post is encrypted for this user or not, it attempts to decrypt every message as if it is intended for this user; that is, it convert the message as if it is Base64-encoded, then decrypt it with the appropriate key (as if it is encrypted for this user). If the Base64 conversion does not result in an IllegalArgumentException and the decryption does not result in a BadPaddingException, it is then assumed to be a correct decryption, and it displays this decrypted message. Otherwise (if one of the exceptions happen), the message is either readable plaintext intended for everyone, or some Base64-encoded string of a message encrypted and intended for someone else; in both cases it displays the *original* message. (Note that it is conceivable that an unencrypted message or a message encrypted for someone else can get past this process without causing these exceptions, but this is unlikely.)
- Note also that this system therefore has this somewhat unusual property: for the intended recipient, the decryption happened "transparently" and they would not know that the message was encrypted and intended only for them; while for all other users they will see the presence of an encrypted message from the sender (although they won't know the recipient).
- After displaying all posts, the client program then asks the user whether they want to post a message. If the user wants to, then it prompts the user to enter the userid of the recipient, and the message. If the user enters "all" as the recipient userid (we assume no one's userid is "all"), then the message is not encrypted. Otherwise, it is encrypted with RSA/ECB/PKCS1Padding and with the appropriate key to ensure only the intended recipient can read it. You can assume the message is short enough so it can be encrypted by RSA in one block. The encryption result is then converted to a Base64 string, and this becomes the "message" part of the post.
- The client program should also generate a signature based on the whole post (the three fields, where the message part is to be treated just like a string whether it was encrypted or not), using the appropriate key to prove the identity of the sender. The post and the signature are then sent to the server.

An example of the client program output may look like this (you do not have to follow the format exactly):

```
There are 2 post(s).

Sender: alice
Date: Wed Jan 26 00:32:19 GMT 2022
Message:
ZpUFbCw3MlnqOzLmTxq2orlhbHxAMVIIzAXHyz3kiuhTzH9xW/RJ6gzCVwkhKA61yx2Uzte/Cvw
Dn8QdaF0WMB9jUtXo1hShp0lFA+tTohPtJxmeELtZEwasM8u0T6YQGP9PhpetgGm9zCnSxfC2
KLV5iXgBr0yOtsmxq+InUqfWH6GNdfaMr+0RTdqS44v00Iz3Eonzj4v0jno+eYjGYsdo1cqKk/bA3I
Zsu8K1kLNZnKdH2W6NomV7uKm5tvzHpWCJkNdyKU4W5mT6NFEj2kFcJ4FKOIQRU7x52Job
1S13/Vc7zbc8dG82FFvWZ8xgBD2kbE4gHh32IJ47nkFI6A==

Sender: bob
Date: Wed Jan 26 00:34:49 GMT 2022
Message: The cake is a lie

Do you want to add a post? [y/n]
y

Enter the recipient userid (type "all" for posting without encryption):
alice

Enter your message:
Hey alice why are you sending some secret message to someone?
```

Submission instructions

Submit **only** the two files Client.java and Server.java. They must be of these exact names (including of the correct upper/lowercase). Just upload them as separate files.