Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

WS 2005-2006

Roger Wattenhofer, Nicolas Burri,
Pascal von Rickenbach, and Yves Weber

Mobile Computing
Exercise 9

Assigned: January 30, 2006
Due: February 6, 2006

1 Shuffle and Deal

In the previous exercise, you designed a distributed algorithm to shuffle and deal the 52 cards of the hearts game. Now it is time to implement such an algorithm. The implementation is placed inside the `HeartsCommunication` class. This class was extended to implement the `HeartsGameActions` interface which offers some higher level commands to the game logic like waiting for the first card, exchanging cards and especially shuffling and dealing. Except the latter, all functions are already implemented in the version available on the course website.

Since all players need to agree on the shuffle algorithm used, you should not implement your own algorithm but use the following specification.

Algorithm Overview

For the shuffling of the cards, we assign special tasks to three of the four players:

- The **Host** starts the algorithm by creating the cards and encrypting them. The (encrypted) cards are sent to **Card Dealer**, the corresponding keys to the **Key Master**[1].
- The **Card Dealer** receives the encrypted cards and randomly distributes them to the other players while keeping 13 cards for himself.
- The **Key Master** gets the keys from the **Host**. It is responsible for distributing the keys. For that task, all other players can ask him for exactly 13 keys. The **Key Master** provides each key (except his own ones) exactly once.

The last player — let's call him **Dummy** during the shuffle and deal process — just waits for this set of encrypted cards and asks the Key Master for his keys.

Why does this algorithm work? The idea is that no single player knows the assignment of the cards and the corresponding keys. The Host knows all keys and encrypted cards, but does not know how they are assigned[2]. The Card Dealer knows this assignment, but only gets the 13 keys he needs for his own cards. The Key Master knows all keys but like the Host, it does not know the assignment of the cards.

[1]When implementing the algorithm, you can save yourself some work if the keys are sent to the Key Master before sending the encrypted cards to the Card Dealer. That way, the Key Master can not receive key requests before he gets the keys. Therefore, he can directly answer the requests and does not need to cache them in some way.

[2]As stated in Exercise 8, we assume that players do not peek at the payload of multi hop messages while forwarding them, i.e. the messages are end-to-end encrypted.

Encryption

To keep the implementation simple and efficient, we do not use serialization of Java objects during the shuffle and deal phase. Each card is represented by a single byte, where the two most significant bits represent the color and the four least significant bits represent the value. The remaining two bits are not used and are set to zero during the whole process.

We use the following encoding for the color (binary) and the values (integer):

- Colors: Clubs (00), Diamonds (01), Spades (10), Hearts (11)
- Values: Ace (1), Values 2 to 10 (2...10), Jack (11), Queen (12), King (13)

To encrypt the byte representing one card, a random byte is generated (the key). This key is then used as One Time Pad, i.e. the card byte is XOR'ed with the key. To decrypt the card, a player simple needs to XOR the (encrypted) card byte with the key again.

Role Assignment

The four players must agree on the assignment of the roles, i.e. who is the Host, who is the Card Dealer, and who is the Key Master. This is done in a very simple way:

When starting a game, the lobby sends a Game object containing an array with the four players to each of them. By definition in the previous exercise, the first player in the array must be the host of the game. We slightly extend this specification by the straightforward requirement that the remaining three players are ordered the same way for all players. This is most probably already the case in your implementation since you (probably) send the same serialized Game object the each player.

We then use the following role assignment: The player at index 0 is the Host, index 1 contains the Key Master, index 2 the Card Dealer, and the last player in the array is the Dummy player.

Message types

We introduce five new message types used by the shuffle and deal algorithm. They are all defined in the HeartsMessage class:

- `SEND_ALL_CARDS`: This message is sent from the Host to the Card Dealer. As payload, it contains a byte array of size 52.
- `SEND_ALL_KEYS`: Similarly as above, this message is sent from the Host to the Key Master and contains the 52 keys. The keys are ordered according to the order of the cards sent as payload of the `SEND_ALL_CARDS` message, i.e. the key at index 7 is required to decrypt the card at index 7.
- `SEND_CARDS`: Messages of this type are sent by the Card Dealer to each of the players to distribute the encrypted cards. It contains a byte array of size 26. The first 13 elements are the cards, the other 13 bytes contain the number of the corresponding key required to decrypt the card.
- `REQUEST_KEYS`: To request a set of 13 keys, a player sends a `REQUEST_KEYS` message to the Key Master. The payload of this message contains a byte array of size 13 that contains the numbers of the keys required.
- `SEND_KEYS`: Upon receipt of a `REQUEST_KEY` message, the Key Master sends a `SEND_KEYS` message back to the source. It contains a byte array containing the 13 keys requested.

2 Play the Game!

Besides the sources available online which are missing the shuffleAndDeal() function, the game is completely available in compiled form on the course website. You should now play some games — either using your own version, our sample solution, or mixing multiple different solutions to check compatibility. The games can be run in the Simulator or (if the hardware cooperates) in the real world using your wireless card in ad-hoc mode.