Steam Key Sharing

# Kevin Ho 100484144, Neil Seward 100487950, Robert cole 100481609

## Abstract

Steam is a PC platform for centralizing games. Games can range from online to offline, and free or paid. Although there is a friends list to allow cooperative play with friends, it requires both friends to have the game. If a friend wants to try a game, he has limited options. He has to either buy the game, play using a friend’s account, or obtain it illegally. There are many issues involved with the methods other than buying the game. The safe solution would be to buy the game. If he does not like the game, then he would have wasted money on a game he would not play and cannot give it away.

With a big platform and the ability to receive a game through its unique steam key, users can install and play the game. These steam keys can be used in many ways. A solution to the problem of sharing games is to transfer over temporary ownership of the steam key for a set amount of time. This transfer can only be done with people on the friends list to keep the key more secure. The temporary owner of the key also cannot transfer ownership of that key to another person to avoid chain reactions and loss of the game.

## Table of contents

[Abstract 2](#_Toc436304365)

[Table of contents 3](#_Toc436304366)

[Project statement 4](#_Toc436304367)

[Requirements 4](#_Toc436304368)

[Preliminary architecture 4](#_Toc436304369)

[Proposed Solution 5](#_Toc436304370)

[Final architecture 5](#_Toc436304371)

[Final Solution 6](#_Toc436304372)

[Demo 6](#_Toc436304373)

## Project statement

Steam lacks the ability to share games with a friend for them to test the game before making a purchase. One user will transfer ownership of the steam key to another user. Once the transfer is completed then the first user will not be able to play the game until key is returned. This project will create a distributed system that facilitates this interaction between users for steam keys. The expected finish date of this project will be November 27th, 2015.

## Requirements

**User Interface**

The project will have a simple interface as a platform of interaction. This interface will allow the user to access the list of games and choose to share them with people in a friends list. A minimalistic but functional user interface will be made only for the interactions within the system.

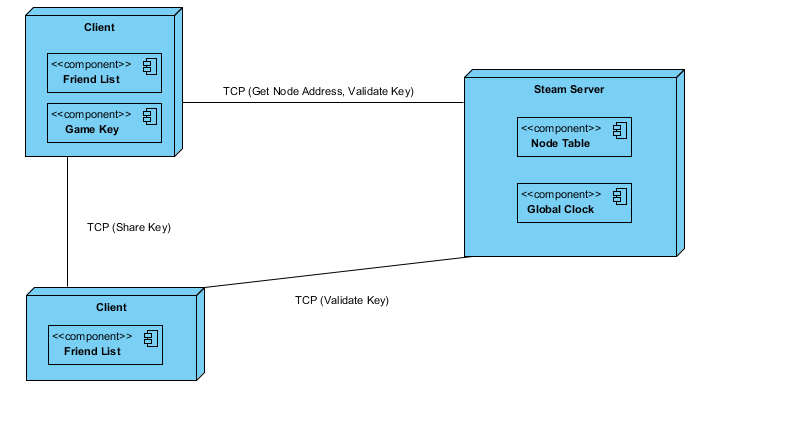
**Objects/Classes**

|  |  |  |
| --- | --- | --- |
| Name | Attributes | Functions |
| User | * List of games * List of users (friends) | * Select game * Select friend |
| Game | * Steam key * Name * Time played | * Share * Play * View details |
| Steam key | * Owner * Current user | * Share * Activate |

**Security**

This project will encrypt the keys to provide safe transfer between users in the system. This is needed to avoid having the keys taken by an external source and used. Because this is a transfer between 2 users, the steam key and the game should only be accessible by the current user of the key.

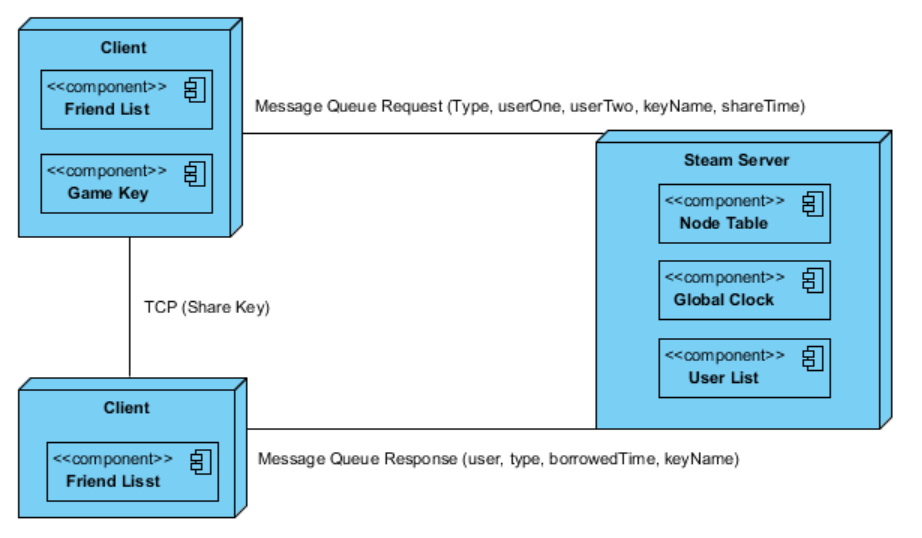
## Preliminary architecture



## Proposed Solution

The interaction between users to the server will be the main connection. These users will be connected through the server to their friends list to determine which users have the ability to share and receive games. The server will not allow games to be shared to users that are not friends with each other. On success, the key will be transferred peer to peer to reduce the load on the server where the server sets the duration of the key on the receiving user side while maintain periodic synchronization for the time. The time will be set from the server using Greenwich Mean Time (GMT) to allow time stamps to be standardized and only translated on user end for better appearance to local time zone. Saved data for the game will persist even after the game is transferred over or back. This allows the game to be shared again and the user to start from where they left off. The data is stored locally in steam and the transfer of keys will just disable or enable access to the saved data.

## Final architecture



## Final Solution

The final solution to the problem follows in the footsteps of the proposed solution. In the final implementation, we chose to use a message queue that all clients would use to communicate to the server. This message queue will contain a series of information: the type of request, the first user, the second user (if it is a sharing request), key name (for sharing request), and share time (for sharing request). The server will then use this information to approve the request or to reject the request. The server contains a list of all the users and is able to confirm if the request can be implemented. Once the request is approved, the server will send the response back to the initial user and if it is a sharing request, it will send a response to the second user as well. Once both users get the response, a TCP link will be set up between the two users to pass the key. The reason for sharing keys in this fashion is to prevent the server to receive more traffic. The server has its own global time that will check for timeouts. When a timeout is detected, the server will send a response to both users involved to release the key and return it back to the owner. This part will involve resetting the key’s values to the original state and setting up another TCP link.

A problem with the architecture is the amount of traffic the message queues will have to the server. This setup is not very scalable with this simple design. A further improvement to increase scalability is to use replication on the server end. This will allow more traffic to flow through easily.

## Demo

For the demo, to simplify the interactions, we decided to only allow the first user to communicate with the server. This will show the interaction of sharing the key, and then showing the updated key. With the use of message queues, to prevent a deadlock where the server will not check for timeouts if there are no messages to retrieve, a timeout of 10 seconds is set on the message queue read portion. If the read is unsuccessful, the server will move on and try again after checking timeouts.

Link: <https://github.com/Robz444/Key_Share.git>

Instructions:

1. Run KeyShareTester.java (no arguments)
2. Enter user 1 name
3. Enter user 2 name
4. Enter ‘SHARE’
5. Enter game name (user 1 name + ’00’) (setup is user1’s name and user1’s id twice ex. Bob00)
6. Enter ‘1’ for the time in minutes
7. Wait for a little more than a minute
8. When able to input again, enter ‘SHOW’