Final Report

Kalie Chang, Zhilin Shi, Oliver Moscow

February 2023

1 Introduction/Motivation

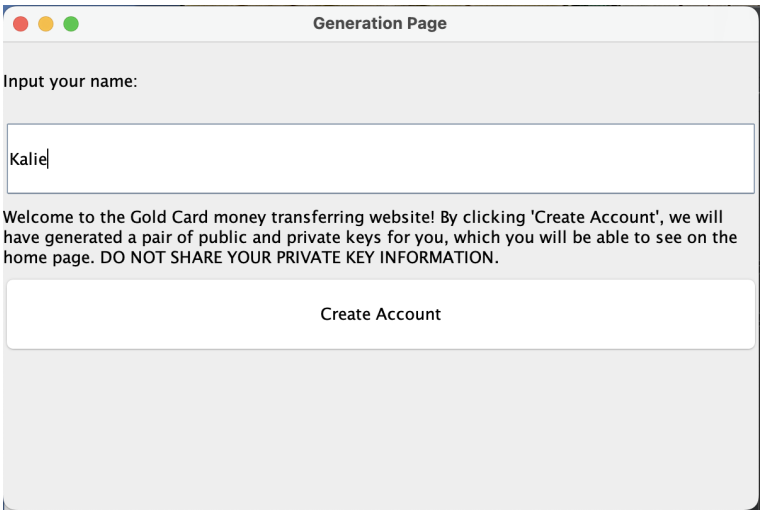
Our group was motivated to create this project since all of us were interested in the concept of public and private key encryption that you brought up during a lesson in class. The initial purpose of the project was to learn how to encrypt something, and then the idea expanded to learning how to do basic encryption on a transaction between two computers through a server. We intend to support the functionalities of generating users with unique public and private keys, storing the user information on a local file so the information isn't deleted when they close out of the program, and transferring information between at least two computers through a server. With the concept and basic functionalities in mind, our group came up with an online ledger and banking system. The purpose of this project is to let users have the ability to transfer money to each other with a public and private key encryption to verify and secure the transaction.

2 Design

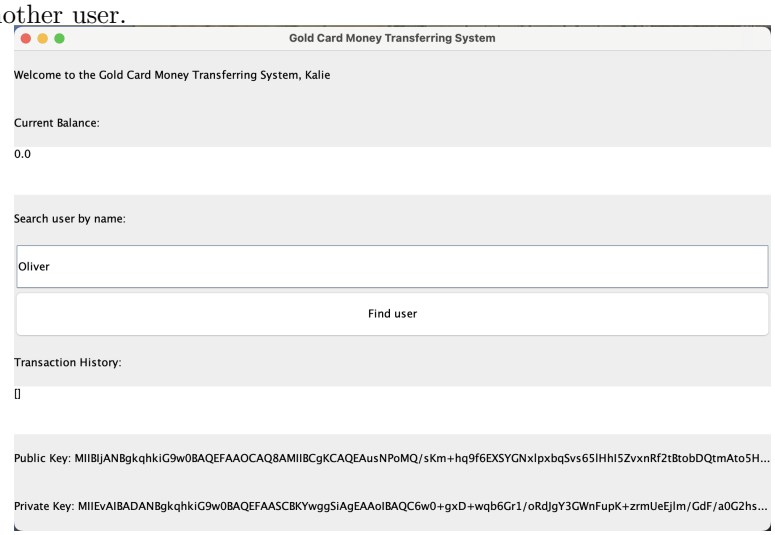
We designed the program with a very clear front end and back end. It supports sending information from the client to the server. The client can engage with the server by sending transactions, viewing their balance, creating their account, and viewing transaction history. The server is able to support encryption of data by checking the public key of the sender with an encrypted message containing the transaction details. The message is encrypted with the sender's private key on the client side.

The idea of encryption is to first have a pair of keys (private key and public key) for our user. When the client requests to send money to a receiver, a message containing the receiver's name and the public key is encrypted with the sender's private key. This encrypted message will be sent to the server along with the sender's public key for decryption. On the server side, transactions are required to have a valid sender and receiver. The transaction will be rejected if: A. the message cannot be decrypted with the sender's public key. B. The sender or receiver is not found within the database. The user can have a negative balance.

On the front end, there is a Window class that serves as the GUI. The Window class has the GUI structures for three different windows. A main functionality of the first window is to store the user's name input as well as generate a pair of private and public keys to store on a file locally on the user's computer. If the user has been initialized, this window will no longer appear again.

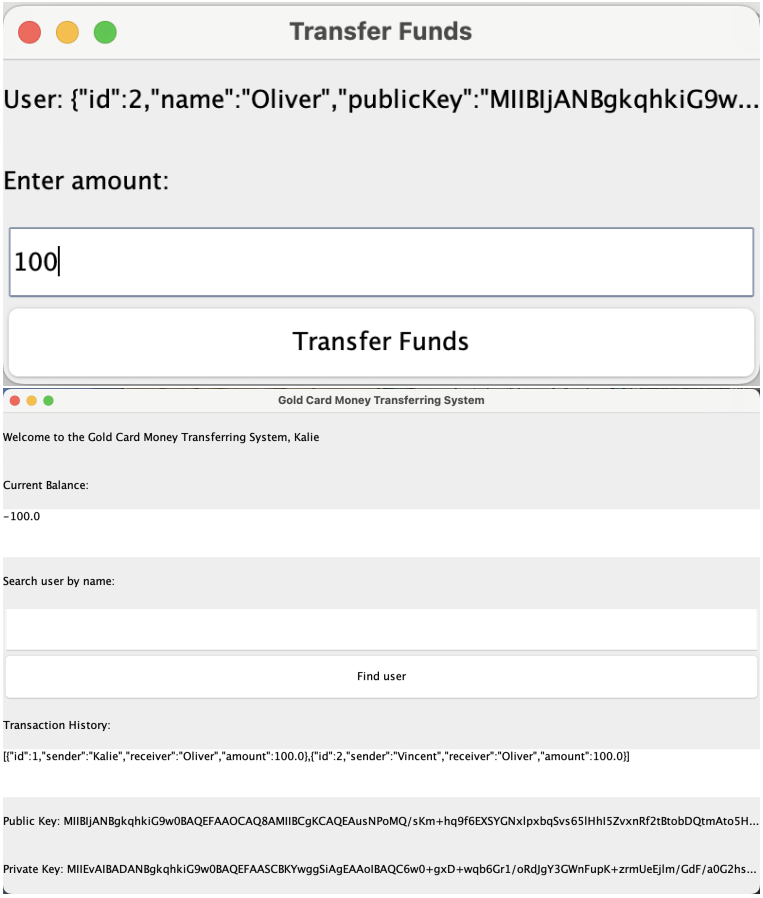


The second window is the main program window, which displays the balance on their account, the transaction history, and allows them to send money to another user.



The third window is a pop up of the user that was inputted into the JTextField and sends information from the user's computer to the server. It will automatically close after send is pressed. If money is sent to the user, the user will be able to see the money through a change in the balance and the transaction

history. They are also able to close the application completely without losing their information since it is stored locally.



The class serverRequest connects the client side with the server side via http requests. Also on the front end is a Backup class, which allows for the user information to be stored on a local text file (which is generated after the user creates their account with the button). Finally, there is a Secret class which handles the generation of RSA key pairs.

On the back end, the project is built via Spring Boot, which creates a JPA Database and Webserver. We are storing two tables in our database, one for transactions (ledger) and one for users. The classes relate through the class Transaction. It defines a default Transaction object and defines a table name and column names for storing the Ledger (List of transactions) in JPA. The class TransactionController has API endpoints that relate to adding and checking transactions. The class Interface TransactionRepository extends a spring boot repository class which connects everything and enables quarrying of the database. The same structure is applied for storing Users: class User, class UserController, class UserRepository. Classes for handling request errors include

class InvalidTransactionExeption, which defines an error for invalid exception and takes in an error message, and class InvalidTransactionAdvance, which creates http response from exceptions. There are also a few helper classes: Class Backup, Backup database, and Class Decrypter, which decrypts the encrypted transaction. The information on the server end can be accessed through a NGROK tunnel which is connected to a local host port on the server's computer. In the future, the server would ideally be deployed on some centralized computers via aws for example.

3 Data

Data is stored on the client end and server end. On the client end, a pair of public and private keys are generated on the first window when the user hits "Create Account". This information is stored on a local file on their computer titled "secret.txt", which will also record the name that the user inputs in the JTextField of the first window.

This local data storage allows the user to keep their private key separate from the server so if there potentially was an issue with the server, like it has been infiltrated for example, the user's information would not be compromised. The client end will be able to access data stored on the server side through http. The transaction history can be seen through the JTextArea at the bottom and the user can access other users by searching them up on the JTextField because the public key and name of the other users are stored on the server side. Users can also access this data through postman or simply by typing AP routes in a

browser.

The data from the server side is stored in a JSON data structure and it is obtained by the client side through requests. The server stores information about the user and the transactions.

```
Recovering... Initialized database with transactions:
com.cc.digitalLedger.Transaction@73ff7a54
com.cc.digitalLedger.Transaction@162e29a1
com.cc.digitalLedger.Transaction@647ce968
Recovering... Initialized database with users:
Kalie
Oliver
Vincent
```

The data can be displayed on the GUI window through a JSON parser. We've considered text files and mapping functions, but we ultimately settled on JSON because it was easier to set up and manage.

4 Issues

A major issue we came across during implementation was to fully integrate the client side and the server side. On the front end, the GUI elements had problems displaying at the correct position that it was set to. Although we weren't quite able to understand why it doesn't display even though we set the location to a specific position, we were able to get the GUI well organized with a grid layout. An issue with instantiating the serverRequest class in the Window class includes the fact that it would automatically open the second JFrame even though the user hasn't created an account. To solve this, we had to restructure the entire Window class into an if-statement so that if the user hasn't created an account that generated the "secret.txt" data file for them, the first JFrame would appear. Else, the second JFrame would appear. This resulted in serverRequest being instantiated twice. The problem was ultimately fixed by setting the serverRequest object to null and instantiating it in the initial GUI.

We also faced some issues with the encryption and decryption algorithms because we had to encrypt messages to an acceptable format which could be sent via HTTP. After lots of research and troubleshooting, we were able to successfully encrypt data and send it to the server.

There were also lots of issues on the backend as we tried to figure out the best way to format the API routes to accept public keys and encrypted methods to be sent through HTTP. We solved these problems in a few different ways. In order to accept encrypted data we are using post requests with the encrypted message included request body. However, we wanted to keep most of the endpoints as get requests so that they could be accessed through the browser and were simpler to implement client side. We ended up including usernames along with each public key. This makes data much more readable as it is easy to see who sends

who money. The username can also easily be sent as a request parameter in a get request.

We also faced some challenges in group work as we needed to make sure all members understood each individual part so we can try to combine everything together. This is because servers are conceptually new for all of us and we all had to do an extensive amount of research to get a good understanding of it.

5 Conclusion

The experience planning the project was optimal because we clearly defined a goal, functions, a front end, and a back end early on, so we didn't have to adjust the idea as we continued to build the project. From a project management perspective, our extensive planning allowed us to fully understand how the project would work from a higher level, and thus allowed us to fully understand the different implementations and features.

From a technical perspective, all of us were exploring the newer and more advanced concept of servers, which we all don't have extensive experience with. Thus, there was a learning curve in understanding the technical aspect of this project, especially on the server end, and how to integrate the server with the client end. For example, a major concept we had to understand was that the client requests information from the server and doesn't care about the methods in the server in the same way that it needs to care about the methods in each class. We also had to understand how to use Ngrok to create a local server and find out what type of data structure best fit with our project.

Overall, we are really pleased with the planning experience as it really highlighted our understanding of how a basic program functions and how different classes interact. Utilizing the server posed difficulties technically and conceptually, but it was a learning experience that we were able to incorporate into fully developing our program. If we had the time, an extension we would add would be a special banking class that allocates a set amount of money for each user and stops them when their money count hits zero. Currently, the user can have negative amounts of money, which means they would owe money to the system, so it would be a notable extension to make the program more realistic.

Check out our source code! https://github.com/comp-sci-2