GFMS: A Group Financial Management System Using Secret Sharing between Server Group

Meng Li
201620728
Department of Computer Science

Yasushi Shinjo
Supervisor
Department of Computer Science

1 Introduction

The conventional applications are based on a client-server mode, which requires a central server for storing shared data. That means the users of the web applications must fully trust central servers and their application providers. Once a server is attacked by hackers, user information may be revealed because data is stored on only one server. Finally, users may lose their data when service providers shut down their services.

Thus, we are going to implement a group financial management application which is not relied on central servers, that means user data cannot be cracked easily and service can be recovered after shutting down the old service. Most current popular way is using P2P(Peer to Peer) to transfer user data between devices. However, there is a obvious problem in such a no-server proposal, that synchronization can only be finished in condition that two devices is online in a same time. Another problem is that the number of P2P connections will be increased fast with the increasing of group scale because each device can create a connection to all of the other devices.

To address this problem without using P2P, we are considering to use multiple servers to build a server group, so that data will divided to several parts and uploaded to different servers. Each server can only save a part of data temporarilywhich means this part will be deleted after a period time we specified. Those two rules explained above can ensure that user data cannot be cracked easily. In fact, we can regard the server group as a bridge between devices of group members. In addition, all devices of group members have saved a complete data set, data can be recovered theoretically even the server group breaks down. By this way, we think such a method can satisfy the requirements for an application which is not relied on central servers.

2 Related Work

related work

3 Methodology

We consider that implementing such a distributed application GFMS on mobile devices is easier than PC, due to the reason that mobile devices can access to Internet by cellular network everywhere and every time. For this reason, we implement our financial management application on iOS mobile devices.

3.1 Data Synchronization by Multiple Servers

We implement GFMS based on data synchronization by multiple servers rather than a central server. The main difference between our proposal and the traditional way using central server is that GFMS is not relying on server. There are three core principles in our proposal.

The first is data transfer by server, not save on server. Most current popular B/S modal application storage user data on several central servers, user data will not be deleted unless user delete his account. GFMS use server as a bridge for transferring data. Considering a group including three members Alice, Bob and Carol. Alice created a new record in her device, this record will be upload to our server group, and will not exist after Bob and Carol synchronized it from server group. In other words, server group storage a record until it is be synchronized by all group members.

The second is server save data uploaded from mobile devices temporarily. We define a period of time in which data can be saved in a server. In this paper, we set this period to 1 hour for our example situations, that means the record Alice uploaded to server group can only be saved for 1 hour. After 1 hour since uploaded, this record will be deleted. Alice is required to resend this record to server group until all of other members has synchronized successfully. It is obvious that this period of time should be long while more members attend the group. However, the longer saving period means the higher risk of data reveal. The most suitable period is influenced by the number of group members, security requirement, user expectations and so on.

The third is server does not know the content of

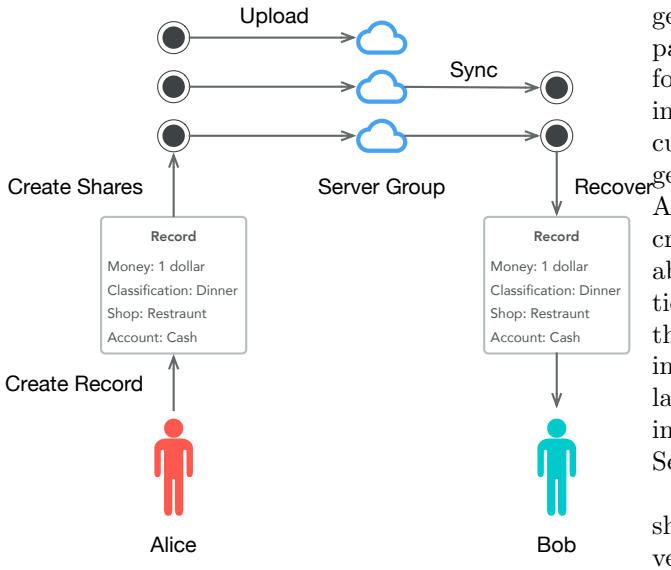

Figure 1: Flow of Synchronization

data. Only by saving data temporarily cannot ensure data security, because server knows the content of data in this temporary period. For this problem, developers usually encrypt data before uploading to servers, which need a private key to decrypt data in other devices. In order to distribute the private key, Sweets, which is a decentralized social networking service application using data synchronization on mobile devices, use QR Code to share the private key. Sweets can generate a QR Code contained the private key, while other devices can scan this QR Code to recover the private key. In this paper, we are going to use secret sharing, which can create a number of shares and distributing them to a set of participants[3], to share data between user devices and server group.

Based on the three core principles introduced above, our synchronization flow can be described as figure 1. In figure 1, Alice create a record including money, classification, shop and account. GFMS create a number of shares by secret sharing and upload those shares to server group. When Bob is online, GFMS in Bob's device will pull data from server group and recover the new record uploaded by Alice. In our situation, GFMS can recover the record after pulling more than two shares randomly. In this process, each server is separated from other server, and cannot access to other server, which means user data cannot be recovered unless you have permission access to more than two server in Alice and Bob's situation. In our proposal, only group members have permission to access server group. Access control will be introduced in the paragraph below.

3.2 Shamir's Secret Sharing

Secret sharing which can create a numbers of shares, plays an indispensable role in protecting user data from getting lost, destroyed, or into wrong hands. In this paper, we are going to use Shamir's secret sharing, a form of secret sharing proposed by Shamir and Blakley independently. In secret sharing scheme, a dealer securely shares a secret with a group of participants, by generating n shares using a cryptographic function[3]. At least k or more participants can reconstruct the secret, but $k-1$ or fewer participants can obtain nothing about the secret[2]. We describe this scheme as a function $f(n,k)$, n is the number of all shares, and k is the threshold to combine shares. A popular technique to implement threshold schemes uses polynomial interpolation ("Lagrange interpolation"). This method was invented by Shamir in 1979. Thus, we call it Shamir's Secret Sharing.

There are many implementations of Shamir's secret sharing by different programming languages. For developing an iOS app by Objective-C, we need an implementation without limitation of length, which supports Objective-C language and UTF-8 character set. C-SSS[1] is an implementation in original C code of Shamir's Secret Sharing by Fletcher T. Penney. C-SSS provide two main functions for generate shares and reconstructing them. Function *generate* can generate n shares by the string text with the threshold k. Function *extract* can recreate the text string after accessing to more than k shares.

```
/**
 Given a text, 'n', and 'k', create
 a list of shares.
 **/
char* generate_share_strings
        (char* text, int n, int t);

/**
 Given a list of shares, recreate
 the original secret.
 **/
char* extract_secret_from_share_strings
        (const char * string);
```

3.3 Synchronize Completely

There is no doubt that GFMS should provide a reliable synchronization service. For Alice, once she created a new record in her device, all of other members in a group should synchronize this record, even if this record may be deleted by server group after 1 hour. We call this problem *Synchronize Completely*.

Figure 2 described the situation data cannot be synchronized completely. Alice sent a new record to server group in 10:00 AM and Bob synchronized it successfully before 11:00 AM. Carol forgot to synchronize this record before 11:00 AM. She cannot synchronize after 11:00 AM because server group has delete the record in 11:00 AM. To solve this problem, Alice is required to resend her new record until Carol synchronize it suc-

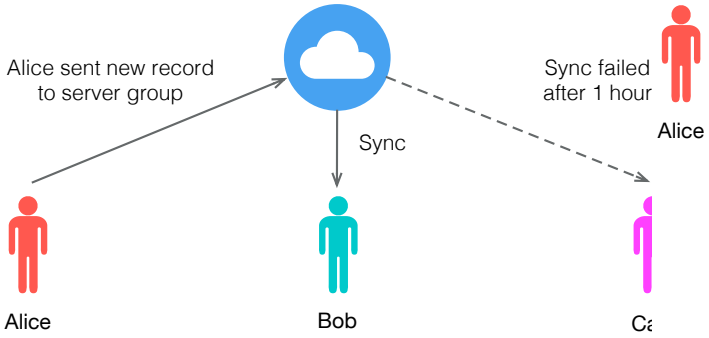

Figure 2: Synchronize Completely

cessfully. However, by what way can Alice get the information that all of other members in her group have synchronized. In other words, it is an unreasonable requests for Alice to resend her record indefinitely. It is necessary for us to find a efficient way for notifying Alice that all members have synchronized successfully.

We are going to design a counter in clients as the solution to *Synchronize Completely* problem. Intuitively, this counter which can calculate the number of times other clients synchronized, works on the sender to decide resending record or not. The condition that sender device need not resend record can be described as equation 1.

$$\sum_{i=1}^{n} s_i = (m-1) \cdot k \tag{1}$$

In equation 1, we suppose that there are n servers in our server group , they are server 1, server 2,..., server n. We can also look at n from a different perspective, it is first parameter, the number of all shares, in the scheme $f(n,k)$ of Shamir's secret sharing. Each server saves a share, so n servers saves n shares. Each server should know how many times the share has been downloaded by clients. Thus, we use s_i to represent the number of times in server i. The sum of s_i is calculated in sender client. In the left of equation 1, m represents the number of group members, and k is the threshold in scheme $f(n,k)$. Obviously, a client can reconstruct data after getting k shares. If $m-1$ clients have reconstructed data, $(m-1) \cdot k$ shares is downloaded. That means clients except sender has synchronized successfully, so sender need not resend.

To implement such a counter, two restrictions about security and connection should be considered. On the one hand, servers cannot communicate with each other. Actually, one server of the group does not know the others exist in the group. On the other hand, clients cannot communicate with each other. Clients do know IP address of other devices, so that they cannot send message with P2P connections. We decide to design a sync table described as figure 3, which is running on both servers and client to statistic sync times.

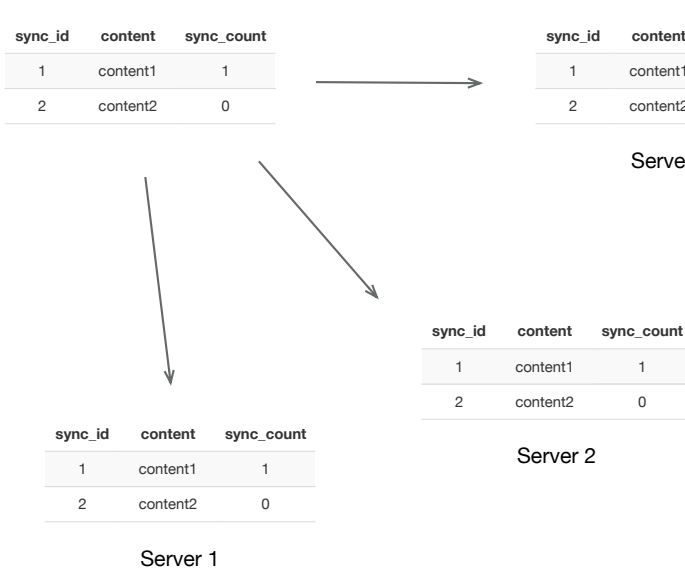

Figure 3: Sync Table

4 Architechture

References

[1] Fletcher T. Penney. *c-SSS, an implementation of Shamir's Secret Sharing*. Version 0.2.0. June 29, 2015. URL: `https://github.com/fletcher/c-sss`.

[2] Liao-Jun Pang and Yu-Min Wang. "A new (t, n) multi-secret sharing scheme based on Shamirfffdfffdfffds secret sharing". In: *Applied Mathematics and Computation* 167.2 (2005), pp. 840–848.

[3] Guillaume Smith, Roksana Boreli, and Mohamed Ali Kaafar. "A Layered Secret Sharing Scheme for Automated Profile Sharing in OSN Groups". In: *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Springer. 2013, pp. 487–499.