Grouper: A Group Finance Manager on Mobile Devices Using Untrusted Servers

Meng Li, 201620728
Supervisor: Yasushi Shinjo

1 Introduction

Conventional mobile applications are based on a client-server mode, which requires central servers for storing shared data. The users of such mobile applications must fully trust central servers and their application providers. Once a server is compromised by hackers, user information may be revealed because data is often stored on the server in cleartext. Finally, users may lose their data when service providers shut down their services. A group finance manager is an application that records the income and expenditure of a small group.

We are implementing a group finance manager application, Grouper, which is not relied on trusted central servers. This means user data cannot be cracked easily and its data can be recovered after services shutting down. A popular proposal is using P2P(Peer to Peer) to transfer user data between devices. However, there is a obvious problem in such a P2P proposal. Data transfer can only be finished during two devices are online at the same time. Another problem is that the number of P2P connections becomes large fast as the number of users' increment.

To address this problem, we use multiple untrusted servers for data transfer. Data will divided to several pieces and uploaded to diverse servers. Each server can only keep a piece of data temporarily. A piece will be deleted after a period time. Those two method can ensure that user data cannot be cracked easily. In addition, all devices of group members keep a complete data set, data can be recovered even untrusted servers shut down.

2 Design

We design a group finance manager application, Grouper, on mobile devices, which is not relied on trusted central servers.

2.1 Group Finance Manager

In Grouper, group members can create a record including money, classification, account, shop, remark and time. With data synchronization, they can also share their records with other group members, so that group income and expenditure information can be analyzed and shown to all members. Grouper uses multiple untrusted servers to synchronize and avoids relying on trusted central servers. To create a group in Grouper, the owner of this group needs to register and set an access key in untrusted servers. The owner passes the access key to group members by a face-to-face way.

2.2 Data Synchronization Using Multiple Untrusted Servers

We implement Grouper based on data synchronization by multiple untrusted servers rather than a single server. There are three principles in our proposal.

Firstly, data transfer by a server, but data does not save on this server permanently. Most current popular client-server applications store user data on several central servers, user data will not be deleted unless user deletes his account. Grouper use untrusted servers as a bridge for transferring data. Consider that a group include three members: Alice, Bob and Carol. Alice created a new record in her device, this record will be upload to untrusted servers, and the record on servers will eliminated after Bob and Carol synchronized it from servers. In other words, untrusted servers stores a record until it is be synchronized by all group members.

Secondly, server keep data 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. This means the record Alice uploaded to untrusted servers can only exist for 1 hour. After 1 hour since uploaded, this record will be deleted. Alice is required to resend this record to serves until all of other members has synchronized successfully. However, the longer keeping 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 others.

Thirdly, servers do not know the content of data. Saving data temporarily cannot ensure data security, because servers know the cleartext of data in this temporary period. For this problem, developers often encrypt data before uploading to servers, which need a private key to decrypt data in other devices. In order to distribute the private key, users should share it

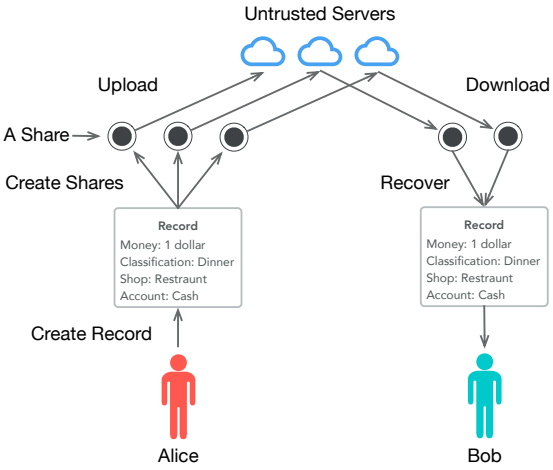

Figure 1: Flow of Synchronization

by themselves. In this paper, we use secret sharing, which can create a number of shares and distributing them to a set of participants[1], to share data between user devices and untrusted servers.

Based on the three core principles introduced above, our synchronization flow can be described as Figure 1. At first, Alice add a record and Grouper creates a number of shares by secret sharing. Then, Grouper uploads those shares to multiple untrusted servers. When Bob is online, Grouper in Bob's device will download shares from servers and recover the new record uploaded by Alice. In this situation, Grouper can recover the record after getting more than two shares. In this process, each server is separated from other servers, and cannot access to other servers. This means user data can not be recovered unless you have permission access to more than two servers. In our proposal, only group members have permission to access these untrusted servers.

2.3 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 a secret sharing scheme, a dealer securely shares a secret with a group of participants, by generating n shares using a cryptographic function[1]. 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 $f(k,n)$, where n is the number of all shares, and k is the threshold to combine shares. Shamir's Secret Sharing is invented by Shamir in 1979 is a popular technique to implement threshold schemes.

In this paper, we use c-SSS[3], an implementation in original C code of Shamir's Secret Sharing by Fletcher T. Penney, which supports UTF-8 character set without limitation of length. It provide two main functions including *generate*, which can generate n shares

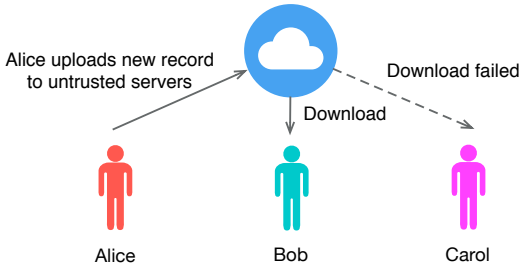

Figure 2: Unreliable Synchronization

by the string text with the threshold k, and *extract*, which can recreate the text string after accessing to more than k shares.

2.4 Reliable Synchronization

Grouper should provide a reliable synchronization service. For example, once Alice, who is a user in a group, creates 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 untrusted servers after 1 hour. We call this problem *Reliable Synchronization*.

Figure 2 describes the situation data cannot be synchronized completely. Alice sent a new record to untrusted servers in 10:00 AM and Bob synchronized it successfully before 11:00 AM. Carol forgot to synchronize, so that she cannot synchronize after 11:00 AM because shares of this record has been deleted by servers. To solve this problem, Alice resend her new record until Carol synchronize it successfully. 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 an efficient way for notifying Alice that all members have synchronized successfully.

We design a counter in clients as the solution to the *Reliable Synchronization* problem. Intuitively, this counter calculates the number of times other clients synchronized and works on the sender to decide resending record or not. The condition that sender device needs not to resend a 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 untrusted servers, Server 1, Server 2,..., Server n. We can also look at n from a different perspective. This is the first parameter, the number of all shares, in the scheme $f(k,n)$ 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 download times in server i. The

Column	Data Type	Introduction
sid	String	Physical unique identifier
content	String	JSON String of object
object	String	Name of object
count	Integer16	Count of sync time
resend	Boolean	Necessity of resending data
sendtime	Date	Data send time

Table 1: Columns of Sender Table

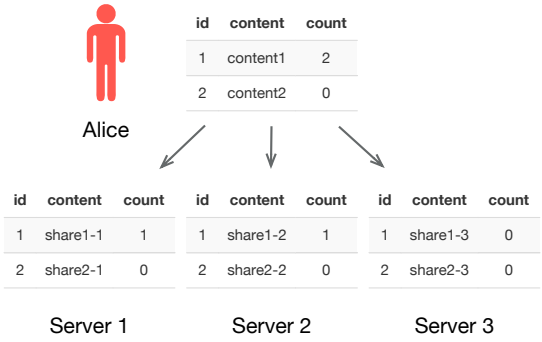

Figure 3: Sender Table and Transfer Table

sum of s_i is calculated in the sender client. In the left of equation 1, m represents the number of group members, and k is the threshold in scheme $f(k, n)$. 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 the sender need not to resend.

To implement such a counter, restriction about security and connection should be considered. Remember that servers cannot communicate with each other. Actually, one server of the group does not know the others in the group. On the other hand, clients cannot communicate with each other. Clients do know IP address or domain name of other device, so that they cannot send messages with P2P connections. We design sender table in clients and transfer table in servers described as Figure 3 to address this problem.

Columns of a sender table are listed in Table 1. Here, sid, content, and count of sync times are main columns. The column sid is an unique identifier created by the SQLite database in a client, it is same as those in servers. The column content saves the JSON string converted by new record object. The column count saves the sync times of this object. In a server, a transfer table also has these 3 main columns with different meaning. The column sid is equals to that in a client in order to point out the row corresponding to that in the client. The column content saves one of the shares generated by secret sharing scheme. The column count saves the number of content sync times in server. Other auxiliary columns in transfer table will not be introduced here.

In Figure 3, suppose that the secret sharing scheme is $f(2, 3)$, and only Bob have synchronized Alice's new record within prescribed time-limit. The column corresponding to Alice's new record, whose id is 1, is equals to those columns whose ids are 1 in Server 1 and Server 2. This means that Bob downloaded two shares from Server 1 and Server 2. The client of Alice will know that by HTTP response from Server 1 and Server 2 when it resend data. It will send for one more time, because it's count is 2, which is less than 4, the threshold Alice needs not to resend.

However, if we consider a further problem that Carol, who is offline, can attack this group by being lazy. Alice, the record sender, has to resend shares to untrusted servers indefinitely. We will address this problem in our future work.

3 Implementation

Grouper consists of an iOS application for clients and a Web service in servers.

3.1 Client

In Methodology, we have introduced how to sharing a string with other deices via multiple untrusted servers. In this section, we first about persistent store and data synchronization. With untrusted servers, Grouper have to store all data on mobile devices with an object-oriented way. Grouper use Core Data[4], a native iOS framework to manage the model layer objects. Core Data provides generalized and automated solutions to common tasks associated with object life cycle and object graph management, including persistence, to store and operate data as object. Sync[5] by Elvis Nuez, a modern Swift JSON synchronization framework to Core Data, can help we implement data synchronization easily in Grouper by parsing a JSON response and getting it into Core Data. With Sync based on Core Date, we can concentrate on sharing with untrusted servers rather than data storage and synchronization.

c-SSS can generate shares from a JSON string and save it into the sender table. Then, Grouper generates shares and send them by a REST API to servers until this record needs not to resend.

3.2 Web Service

Many applications with synchronization is based on commercial cloud services like Amazon S3 or Google Cloud. It is necessary that Grouper has its own Web service running on an individual server rather than using commercial cloud services directely for following reasons:

- Data Storage: untrusted servers save share generated by secret sharing.

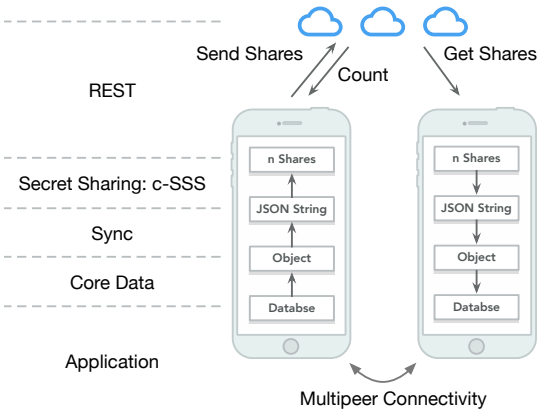

Figure 4: Architecture of Client

- Provisionality: the web service ensure that a share can be delete after a prescriptive time.
- Security: only group members who has an access key can download shares from servers. In other word, a web service plays a role of access control.
- Counter: web service palys a role of a counter to calculate the number of sync times in servers.

4 Related Work

DepSky[6] is a system that stores encrypted data on servers and runs application logic on the client-side[7]. DepSky provide a storage service that improves the availability and confidentiality provided by commercial storage cloud services. *Cloud-of-Clouds* is the core concept in DepSky. It represents that DepSky is a virtual storage cloud, which is accessed by its users by invoking operations in several individual severs. DepSky keeps encrypted data in commercial storage cloud services and do application logic in individual servers. In Grouper, untrusted servers undertake responsibility of temporarily data storage and message delivery with server-side computation.

Mylar[8] stores encrypted and sensitive data on the server, and decrypts that data only in users browsers. Developers of Mylar use its API to encrypt a regular (non-encrypted) Web application, and users decrypt data by a browser extension. Like Grouper, application in Mylar can control how user data is shared[7]; unlike Grouper, Mylar builds its system on a browser with browser extension while Grouper uses mobile device as a client.

There are many applications or frameworks synchronize with untrusted network and servers. Compared to them, Grouper uses secret sharing and temporary data storage with untrusted servers to protect user data from malicious attacking. It is convenient and fast compared to those applications or frameworks using data encryption.

5 Future Work

We are developing an application running in a iOS device and a web service running on Tomcat server. In future, we are going to connect clients and multiple untrusted servers by REST API to test synchronization, and evaluate this application.

6 Conclusion

This paper introduce Grouper, a group finance manager synchronized with multiple untrusted servers, using secret sharing and data synchronization on mobile devices. Grouper consist of an iOS application and a Web service. Each server of Grouper does not know the others and saves one of the shares generated by secret sharing temporarily, to ensure that user data cannot be cracked easily.

References

[1] 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*, pages 487–499. Springer, 2013.

[2] Liao-Jun Pang and Yu-Min Wang. A new (t, n) multi-secret sharing scheme based on shamirs secret sharing. *Applied Mathematics and Computation*, 167(2):840–848, 2005.

[3] Fletcher T. Penney. c-SSS, an implementation of Shamir's Secret Sharing, https://github.com/fletcher/c-sss.

[4] Apple Inc. Core Data Programming Guide, https://developer.apple.com/library/content/documentation/cocoa/conceptual/coredata/.

[5] Elvis Nu nez. Sync, https://github.com/SyncDB/Sync.

[6] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4):12, 2013.

[7] Frank Wang, James Mickens, Nickolai Zeldovich, and Vinod Vaikuntanathan. Sieve: cryptographically enforced access control for user data in untrusted clouds. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 611–626, 2016.

[8] Raluca Ada Popa, Emily Stark, Steven Valdez, Jonas Helfer, Nickolai Zeldovich, and Hari Balakrishnan. Building web applications on top of encrypted data using mylar. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 157–172, 2014.