Grouper: A Framework for Developing Mobile Application Using Secret Sharing and Untrusted Servers

Meng Li, 201620728
Supervisor: Yasushi Shinjo

July 11th, 2017

1 Introduction

Conventional client-server mode applications requires central servers for storing shared data. The users of such mobile applications must fully trust the central server and their application providers. If a central server is compromised by hackers, user information may be revealed because data is often stored on the server in cleartext. Users may lose their data when service providers shut down their services.

To solve this problem, some applications use P2P(Peer to Peer) to transfer user data between devices. Some applications encrypt and decrypted user data in clients and save encrypted data in servers. Unlike them, we use multiple untrusted servers for data transfer. Data is divided into several pieces by Secret Sharing scheme and uploaded to diverse servers. Each server can only keep a piece of data temporarily, we call it share. A share will be deleted after a period of time. These ensure that user data cannot be cracked easily. In addition, all devices of group members keep a complete data set, and data can be recovered even untrusted servers shut down.

2 Design

We design this framework on iOS platform at first. We are aiming at developing applications which is not relying on trusted central server using our Grouper framework.

2.1 Overview

Applications by Grouper can synchronize data between members of a group. In our situation, each group member holds a device. Thus, Grouper should provides main functions.

- Data synchronization: the group member create an object and others synchronize this object.
- Group management: the group owner create a group in his device and invites other members to his group.

2.2 Data Synchronization

In Grouper, a device use Secret Sharing scheme to divide a message into several shares and upload them to multiple untrusted servers. Other devices download shares and recover shares to original message by Secret Sharing scheme.

2.2.1 Shamir's Secret Sharing

Secret sharing plays an indispensable role in protecting user data from getting lost or destroyed. In a secret sharing scheme, a dealer securely shares a secret with a group of members by generating n shares using a cryptographic function[2]. At least k or more shares can reconstruct the secret, but $k-1$ or fewer shares can obtain nothing about the secret[3]. 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. We use Shamir's secret sharing that is a popular technique to implement threshold schemes.

2.2.2 Multiple Untrusted Servers

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

Firstly, a server transfers data as similar to a router, but does not keep it permanently. Most current popular client-server applications store their user data on several central servers, and the user's data will not be deleted unless the user deletes his account. Grouper uses untrusted servers as a bridge for transferring data. Consider that a group includes three members: Alice, Bob and Carol. Alice creates a new record in her device, this record is uploaded to untrusted servers, and the record on servers will be deleted after Bob and Carol download it from servers.

Secondly, a server keeps data temporarily. We define a period of time in which data can be kept in a server, we call it interval time. In this paper, we set this period to 1 hour for our example situations. The record Alice uploaded to untrusted servers can exist for 1 hour. After 1 hour since uploaded, this record

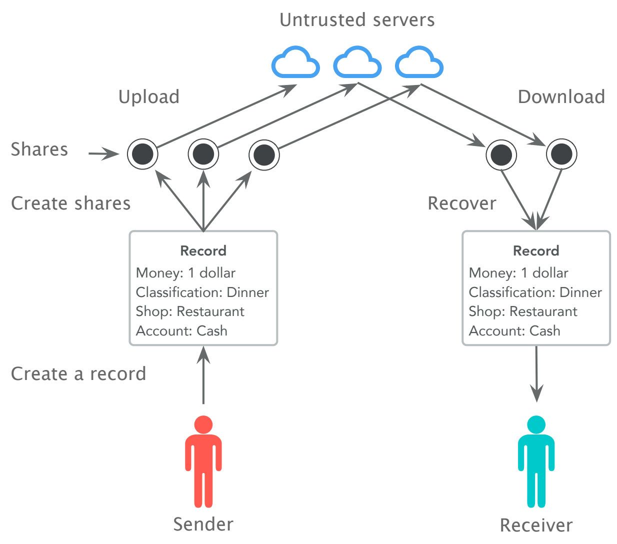

Figure 1: Flow of synchronization.

will be deleted. Alice requires to upload this record again to serves until all of other members have downloaded successfully. The longer keeping period means the higher risk of data reveal. We will find suitable periods as the number of group members, security requirement and others.

Thirdly, servers do not know the cleartext of data. Keeping 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, and this needs a decryption key to decrypt data in other devices. In order to distribute the decryption key, users should share it by themselves. In Grouper, we do not encrypt data but we use a secret sharing scheme as described in Section 2.2. Servers do not know the cleartext of a share generated by a secret sharing scheme.

Figure 1 describes our synchronization flow based on these three principles. At first, Alice adds a record and Grouper creates three shares by a secret sharing scheme where n is three and k is two. Next, Grouper uploads those shares to three untrusted servers. When Bob is online, Grouper in Bob's device downloads two shares from servers and recovers the new record created by Alice. In this situation, Grouper can recover the record after getting two or more shares. In this process, each server is separated from other servers, and cannot access to other servers. This means these servers cannot recover user data because they do not have permission to access other untrusted servers. In our proposal, only group members have permission to access these untrusted servers.

2.3 Group Management

2.3.1 Creating Group

A group is created by its owner. Before creating a group, the owner should prepare his own user information including email and name, and add multiple untrusted servers by submitting group ID and group name. Here, the group ID and group name are assigned by Alice and they should be same in all untrusted servers of this group. Then, he can initialize this group on all untrusted server by submitting his node identifier. The node identifier which can represent his device, is generated by Grouper framework randomly when the application is launched at first time. In each untrusted server, the Web service initializes this new group and returns a master key including the highest privilege to the owner. The owner can add other members to an untrusted server by the master key.

2.3.2 Inviting Member

After creating a group, the owner can invite a new member to his group. To join the group, the new member should also prepare his user information at first. The owner can only invite the new member by a face-to-face way rather than connection via Internet using central server. Before inviting, Grouper establish connection between their devices. Firstly, the new member sends user information and node identifier to the owner. Owner saves user information and node identifier of the new member to his device. Secondly, the owner register the new member on multiple untrusted servers by submitting node identifier of the new member. Thirdly, untrusted servers returns access key for the new member to the owner. Lastly, the owner send access key, server related information and all users' information he has to the new member. After receiving them, the new member can access untrusted servers.

2.4 Reliable Synchronization

Grouper should provide a reliable synchronization service. For example, Alice is a user in a group and 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 a period of time. We call this problem *reliable synchronization.*

Figure 2 describes an unreliable situation that data cannot be synchronized. Alice sends a new record to untrusted servers and Bob downloads it successfully within the interval time. Carol becomes online after the interval time and fails to download shares of this record because they have been deleted by servers. To solve this problem, Alice should upload her new record again until Carol downloads it successfully.

To solve this problem, we reference the basic reliable multicasting scheme in distribute systems. Messages in this scheme should have sequence number which is added one by one, to indicate the sending order of those messages. For example, the sender of a group send a message No.25 and all of the receivers receive the message No.25 successfully. When the receivers receive new message No.25, they will compare No.25 with the newest sequence number in their persistent store. One of the receivers finds that his newest sequence number is No.23, that means he missed the message No.24. Thus, when they send feedback to the receiver, he will report that he missed the message No.24. At last, the sender will send the missed message to him.

In Grouper, we use such solution to solve to reliable synchronization problem. To let the receiver receives all messages, the sender should confirm with the receiver. In figure 2, the sender has sent message from No.1 to No.4. At first, he ends sequence numbers of all messages he sent to the receiver. When the receiver receives the sequence numbers, he will check his local persistent. In this situation, the receiver will find that he missed the message No.3 and No.4. Thus, he will report a feedback that contains the sequence numbers of messages he missed to the sender. At last, the sender will resend the message No.3 and No.4 to the receiver again.

2.5 Grouper Message

To transfer data between devices, we reference the protocol of email and design our own *Grouper Message*.

In fact, a Grouper message is a JSON string that contains the sync entity related information and the way to handle it in receivers device. There are 4 types of Grouper message, they are update message, delete message, confirm message and resend message. Both update message and delete message need resend because they contain sync entity related information. We call them normal message. Both confirm message and resend message contain control information about reliable multicast and need not resend. We call them control message.

When a user creates a new entity or modifies some attributes of an existed entity, the application should send an update message that contains the JSON string of this entity to all group members.

When a user deletes an existed entity, the application should send a delete message that contains the object ID of this entity to all group members.

For reliable multicasting, when the application is launched, it should check the send time if the last confirm message. If it sent before the interval time, the application should send a new confirm message that contains the sequences of all messages which are sent by this devices before, to all group members.

When the device of a group member receives a confirm message, it should check the sequences. If some of them are nonexistent in its' persistent store, the application should send a resend message that contains the nonexistent part of the sequences.

3 Implementation

Grouper consists of a client framework for developing iOS application and a Web service running on multiple untrusted servers. In Section 2, we have introduced how to share strings with other deices via multiple untrusted servers. In this section, we describe persistent store and data synchronization. Grouper stores all data on mobile devices with an object-oriented way.

3.1 Web Service

Grouper needs its own Web service rather than using commercial general cloud services like Amazon S3, Google Cloud for following reasons:

- The Web service supports reliable synchronization.
- The Web service ensures that shares are deleted after a prescriptive time.
- The Web service allows only group members who have access keys to download shares.

Our Web service runs on the Tomcat server that is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. We use the Spring MVC, a Web model-view-controller framework, to create our RESTful API, and Hibernate, an open source Java ORM framework, to save and operate object in the Web service.

Our Web service include three entities. They are *Group*, *User* and *Transfer*. The *Group* entity saves the unique identifier, group name and its owner. The *User* entity save the node identifier of a user, access key for this user, and group of this user. The *Transfer* entity saves a share generated with a secret sharing scheme, the time when the user uploads it and its creator. For each user, there is a unique access key for him in an untrusted server. Thus, only this user can modify or remove what he uploaded to this server. For a group, one of a user is its owner who has the highest privilege of this group.

In a word, our web service provides RESTful API to transfer data with clients, and it keeps shares from devices temporarily.

3.2 Client

Grouper provides a client framework by Objective-C language to developing applications on iOS, macOS, watchOS and tvOS.

Figure 4 describes the architecture of client framework. It is based on such frameworks.

- Grouper used *Multipeer Connectivity*[4], a official peer-to-peer communication framework provided by Apple, to transfer data between devices by a face-to-face way.
- Grouper uses *Core Data*[5], a official ORM framework provided by Apple, 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.
- Grouper uses *Sync*[6], a modern JSON synchronization framework for *Core Data*, to create JSON strings and synchronize object from JSON strings.
- Grouper uses *c-SSS*[7], an implementation of Shamir's secret sharing in the C language, to create shares and recover shares.
- Grouper uses *AFNetworking*[8], a delightful networking library by Objective-C language, to revoke the RESTful API provided by our Web services.

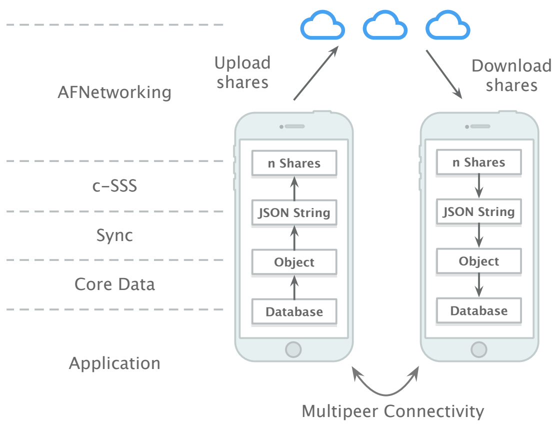

Figure 2: Architecture of the iOS application.

4 Evaluation

This section answers two main questions: first, how much developer effort is required to use Grouper, and second, what are the performance overheads of Grouper?

4.1 Developer Effort

There are two factors that will influence the developer effort: the usability of client API and the lines of code(LoC) the developer has to add.

Client API Grouper provides very simple client API for developers.

```
// Initialize Grouper
grouper.setup(withAppId: "test",
dataStack: DataStack(modelName: "Example"))
// Create Object & Update Object
grouper.sender.update(test)
// Delete Object
grouper.sender.delete(test)
// Receive Object
grouper.receiver.receive {
// Callback after receiving objects.
}
//Send Confirm Message
grouper.sender.confirm()
```

LoC We developed 2 applications including *Account Book* and *Test* with Grouper. As described in table 1, compared with application LoC, Grouper related LoC that developer has to add is only a small part.

Table 1: Demo app's lines of code

Attribute	Test	Account Book
Platform	iOS	iOS
Lanaguage	Swift	Objective-C
Application LoC	632	8950
Grouper related LoC	11	19
Number of Entities	1	5

4.2 Performance

The performance goal is to avoid significantly affecting the user experience with the application developed with Grouper. To evaluate whether Grouper meets this goal, we answer the following questions:

Data Sending: How much processing time does Grouper add to app when a user creates objects? Figure 3 shows the processing time for sending and receiving 1 message. Compared to network, data sync and secret sharing consume very little time in both data sending and receiving.

Data Sending: How much processing time does Grouper add to app when a user wants to synchronize data from untrusted servers? Figure 4 shows the processing time for sending multiple messages. With the increasing of messages number, data sync and Secret sharing part increase linearly. The network part increases very slowly and sometimes decrease.

5 Related Work

DepSky[9] is a system that stores encrypted data on servers and runs application logic. DepSky provides

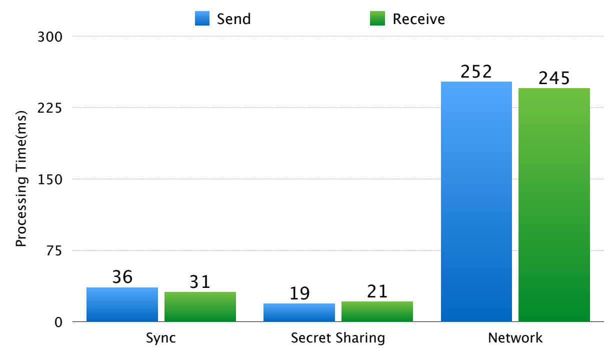

Figure 3: Processing time of sending and receiving 1 message

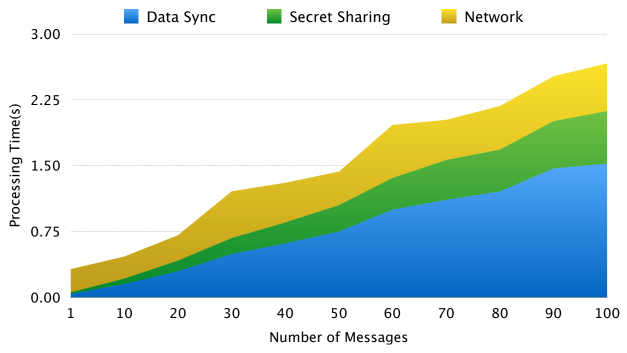

Figure 4: Processing time of sending multiple messages

a storage service that improves the availability and confidentiality by using commercial storage services. *Cloud-of-Clouds* is the core concept in DepSky. It represents that DepSky is a virtual storage, and its users invoke operations in several individual severs. DepSky keeps encrypted data in commercial storage 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[1] stores encrypted and sensitive data on a server, and decrypts this 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 in Grouper, applications in Mylar can control how user data is shared. Mylar builds its system on a browser with extension while Grouper uses mobile devices.

There are many applications and frameworks that use untrusted networks and servers. Compared to them, Grouper uses secret sharing and temporary data storage with untrusted servers to protect user data. Using secret sharing is more convenient and faster than using data encryption because clients need not to distribute decryption keys and perform heavy encryption computation.

6 Conclusion

This paper introduces Grouper, a group finance manager that synchronizes data among mobile devices with multiple untrusted servers. Grouper uses secret sharing and temporary data storage services. Grouper consists of clients that run an iOS application and multiple untrusted servers that run a Web service. Each server of Grouper does not know the others and keeps one piece of the shares generated by secret sharing temporarily to protect user data. We connect clients and multiple untrusted servers by a REST API. We are developing a robust protocol of reliable synchronization using multiple untrusted servers.

References

[1] 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.

[2] 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.

[3] 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.

[4] Apple Inc. MultipeerConnectivity. `https://developer.apple.com/documentation/multipeerconnectivity`.

[5] Apple Inc. Core Data Programming Guide, Guides and Sample Code. `https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/`.

[6] Elvis Nuněz. Sync, modern Swift JSON synchronization to Core Data. `https://github.com/SyncDB/Sync`.

[7] Fletcher T. Penney. c-SSS, an implementation of Shamir's secret sharing. `https://github.com/fletcher/c-sss`.

[8] AFNetworking, a delightful networking framework for iOS, OS X, watchOS, and tvOS. `http://afnetworking.com`.

[9] 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.