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 mobile applications are built based on a client-server mode and require the central servers for storing shared data. The users of such mobile applications must fully trust the central servers. If central servers are 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 servers.

To address these problems, we are developing Grouper, a framework for developing mobile applications. Grouper uses multiple untrusted servers for data transfer. Data is divided into several pieces by a Secret Sharing scheme and uploaded to those untrusted servers. Each server can only keep a piece of data called a 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 of this group, and data can be recovered even untrusted servers shut down.

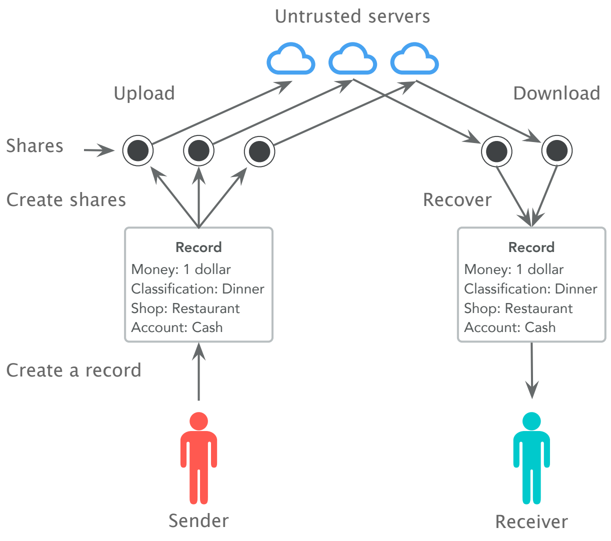

Figure 1: Data Transportation Flow

2 Design of Grouper

2.1 Overview

We are aiming at developing applications which are not relying on trusted central servers. The Grouper framework provides the following functions to applications.

- Data Synchronization: If an user updates an object in his device, the mirror of this object in other devices are updated.
- Group management: The group owner can create a group and invite other members to his group.

We implement data synchronization using the Sync framework[1] and our own messaging function called *Grouper Message*. Using the Sync framework, we get a JSON string from an updated object, send the JSON to other devices, and update the mirror of the object in these devices. We implement *Grouper Message* using untrusted servers (Section 2.2) and a Secret Sharing (Section 2.3) scheme . Untrusted servers delete messages after a period of time. *Grouper Message* implements reliable messaging over this feature (Section 2.3).

2.2 Threat Model

We design Grouper based on data synchronization through multiple untrusted servers. Our untrusted servers have the following features.

Firstly, servers must perform device authentication. Servers generate access keys for group users. When a device want to get and put data to untrusted servers, the device sends a request with an access key in the request header.

Secondly, servers keep data temporarily. We define a period of time in which data can be kept in a server as an interval time. The data in untrusted servers vanishes after the interval time.

In members inviting, group owners authenticate group members by a face-to-face way. We assume that inter-mobile device communication is secure. All group members must trust the group owner and they are not malicious.

2.3 Data Synchronization

2.3.1 Data Transportation

Figure 1 describes our data transportation flow using multiple untrusted servers. At first, the sender adds a record and Grouper creates three shares by a secret sharing scheme. In Grouper, we use a Secret Sharing scheme to divide a message into several shares and upload them to multiple untrusted servers. In a Secret Sharing scheme, a member securely shares a secret with a group of members by generating n shares using a cryptographic function. At

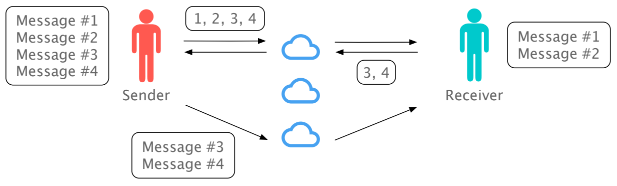

Figure 2: Reliable Synchronization

least k or more shares can reconstruct the secret, but $k-1$ or fewer shares 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.

Next, Grouper uploads those shares to three untrusted servers. When the receiver is online, he downloads two shares from two servers and recovers the new record. 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.

2.3.2 Reliable Synchronization

Grouper should provide a reliable synchronization service. A user in a group creates a new record and all of other members in this group should synchronize this record, even if this record may be deleted by untrusted servers after an interval time. We call this problem reliable synchronization. A receiver can only download shares from untrusted server with the interval time. If he tries to download data after the interval time, he will miss the new record.

To solve this problem, we use the idea of reliable multicasting in distribute systems. In Figure 2, the sender has sent messages from No.1 to No.4. Next, he sends sequence numbers of all messages he sent to the receiver. When the receiver receives the sequence numbers, he checks his local persistent store. In this situation, the receiver finds that he missed the message No.3 and No.4. Thus, he will send a request 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.

2.3.3 Grouper Message

To transfer data between devices, we design our own protocol, *Grouper Message*. A Grouper message is a JSON string that contains an object of an application and the way to handle it in receivers devices. There are 4 types of Grouper messages: update message, delete message, confirm message and resend message. Both update message and delete message need resend because they contain the objects of an application. We call these messages normal messages. Both confirm message and resend message contain control information about reliable multicast and need not resend. We call these messages control messages.

When a user creates a new object or modifies some attributes of an existing object, the device sends an update message that contains the JSON string of this entity to all group members.

When a user deletes an existing object, the device sends a delete message that contains the object ID of this object to all group members.

To confirm all group members have received all normal messages created by a user, the device of him sends a confirm message to group members periodically. In Grouper, when the application is launched, it should check the last sent time of the confirm message. If the last confirm message is sent within interval time, the device need not to send a confirm message again. Otherwise, the devices need to send a new confirm message contains the sequences of all normal messages created by this user before.

When the device of a group member receives a confirm message, it checks the sequence numbers. If some of them do not exist in the persistent store, the device sends a resend message that contains missing sequence numbers.

2.4 Group Management

2.4.1 Creating a Group

A user creates a group, and he becomes the owner of this group. Before creating a group, the owner prepares his own user information including his email and name, multiple untrusted servers, a group ID and a group name. For example, Alice assigns a group ID and a group name and registers them to all untrusted servers of this group. Next, she initializes this group on all untrusted server by submitting her node identifier. The node identifier, which represents her device, is generated by Grouper randomly when the application is launched at the 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.

2.4.2 Inviting a Member

After creating a group, the owner can invite a new member to his group. To join the group, the new member prepares his user information at first. The owner invites the new member by a face-to-face way rather than using central servers. Before inviting, Grouper establishes connection between their devices using local secure link provided by *Multipeer Connectivity*[3]. Firstly, the new member sends user information and a node identifier to the owner. Owner saves the user information and the node identifier of the new member to his device. Secondly, the owner registers the new member on multiple untrusted servers by submitting the node identifier of the new member. Thirdly, untrusted servers returns access keys for the new member to the owner. Lastly, the owner sends the access keys, server addresses and the list of existing members to the new member. After receiving them, the new member can access these untrusted servers with the keys.

3 Implementation

Grouper consists of a Web service (Section 3.1) running on multiple untrusted servers and a client framework (Section 3.2) for developing applications .

3.1 Web Service

Grouper needs its own Web service rather than using commercial general cloud services like Amazon S3, Google

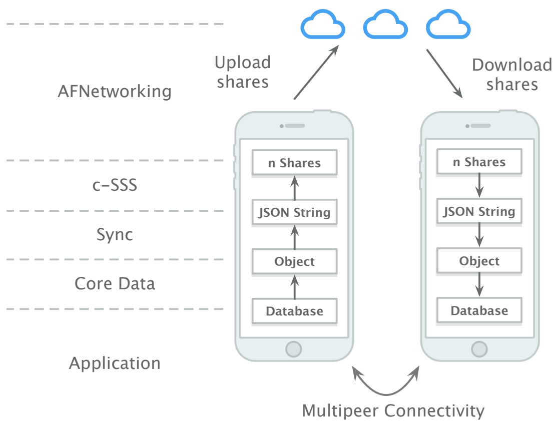

Figure 3: Architecture of Client

Cloud for the following reasons:

- The Web service must provide reliable synchronization based on the *Grouper Message* protocol.
- The Web service must ensure that shares are deleted after a prescriptive time.

Our Web service provides RESTful API to transfer data with clients. It 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 objects in the Web service.

Our Web service includes three kinds of entities. They are *Group*, *User* and *Transfer* entities. A *Group* entity saves a group ID, a group name and its owner. A *User* entity saves the node identifier of a user, the access key for this user, and a group of this user. A *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. For a group, one of a user is its owner who has the highest privilege of this group.

3.2 Client

Grouper's client framework is developed in Objective-C, and it supports developing applications on iOS, macOS, watchOS and tvOS . Figure 3 describes the architecture of client framework. It is based on the following frameworks.

- *Multipeer Connectivity*[3], an official Peer-to-Peer communication framework provided by Apple. Grouper uses it to transfer data between two devices by a face-to-face way.
- *Core Data*[4], an official ORM framework provided by Apple. *Core Data* provides generalized and automated solutions to common tasks associated with object life cycles and object graph management, including persistence. Grouper uses it to manage model layer objects.
- *Sync*[1], a synchronization framework for *Core Data* using JSON. When a user sends messages, Grouper uses it to create JSON strings from objects. When an other user receives messages, Grouper use it to parse JSON string and synchronize the recovered objects into *Core Data*.
- *c-SSS*[5], an implementation of the Secret Sharing scheme.
- *AFNetworking*[6], a delightful networking library in Objective-C. Grouper uses it to invoke the RESTful API provided by our Web services running on multiple untrusted servers.

3.3 Applications

Using Grouper framework, we are developing the following applications.

- *Account Book*, an iOS app in Objective-C, records the income and expenditure.
- *Test*, a benchmark iOS app in Swift, tests the performance of Grouper.
- *Notes*, a macOS app in Swift, takes shares notes for small groups.

4 Evaluation

This section shows the developer effort to use Grouper and the performance of Grouper.

4.1 Developer Effort

We see developer efforts through two factors: the usability of the client API andthe code size of the lines of code(LoC) the developer has to add after using Grouper. Grouper provides the following simple client API for developers.

```
// Setup Grouper with appId and dataStack.
grouper.setup(withAppId:"id",
  dataStack:DataStack(modelName: "name"))
// Update an object.
grouper.sender.update(object)
// Delete an object.
grouper.sender.delete(object)
// Receive objects.
grouper.receiver.receive {
  // Callback after receiving objects.
}
// Send a confirm message.
grouper.sender.confirm()
```

We have developed two applications including *Account Book* and *Test* with Grouper. As described in Table 1, based on the standard application without data synchronization, developers can develop these applications with Grouper by adding a small number of code.

Table 1: Applications' lines of code

Application Name	Test	Account Book
Platform	iOS	iOS
Lanaguage	Swift	Objective-C
Number of Entities	1	5
Stand Alone Application LoC	621	8760
Application with Groper LoC	632	8950
Increased LoC	11	19

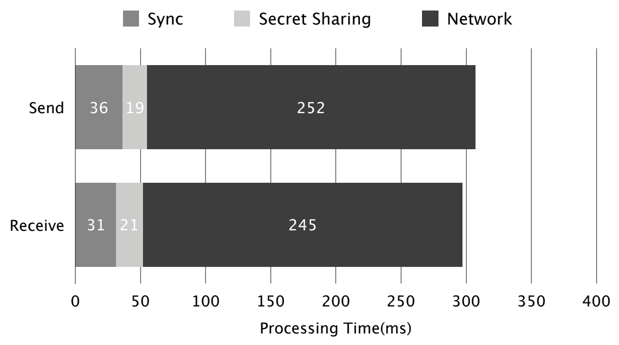

Figure 4: Sending and Receiving a Single Message

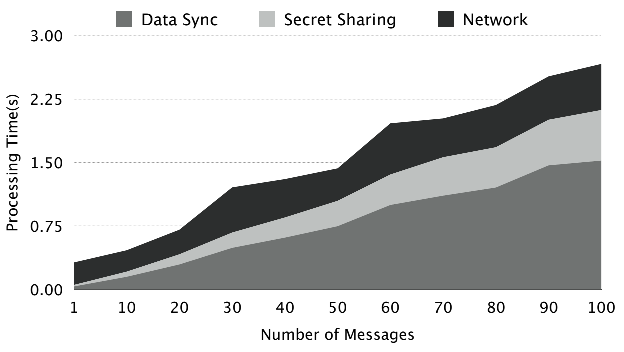

Figure 5: Sending Multiple Messages

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 use the benchmark app *Test* with the $f(2,3)$ Secret Sharing scheme to transfer data between iPhone 4s and iPod 5 generation on a wireless LAN network (802.11n). Both iPhone 4s and iPod 5 generation use A5 CPU and 512MB RAM.

First, we measured processing times the benchmark app *Test* needs when a user creates a test object during online. We use iPhone 4s to create an object and send a update message. We use iPod 5 generation to receive the update message from iPhone 4s. Figure 4 shows the processing time for sending and receiving a single message after creating this object. Compared to the network time, data sync and secret sharing consumed very little time in both data sending and receiving.

Next, we measured processing times *Test* needs when a user becomes online and synchronizes data from untrusted servers. We use iPod 4s to create multiple objects and send multiple update messages at the same time. Figure 5 shows the processing time for sending multiple messages. As the number of messages increased, data sync and Secret sharing part increased linearly. The network part increased very slowly and sometimes decreased.

5 Related Work

To solve the problem using central servers, some applications use Peer-to-Peer (P2P) models to transfer user data among devices. Some applications encrypt and decrypted user data in clients and save encrypted data in servers.

Mylar[7] stores encrypted data on servers, and decrypts this data only in users browsers. Developers of Mylar use its API to encrypt a regular (non-encrypted) Web application. Mylar uses its browser extension to decrypt data on clients. Like in Grouper, applications in Mylar can control how user data is shared and stored on multiple untrusted servers. Mylar builds its system on a browser with extension while Grouper builds on mobile devices.

Sweets[8] is a decentralized social networking service (SNS) application using data synchronization with P2P connections among mobile devices. Sweets uses Advanced Encryption Standard (AES) to encrypt user data and Attribute Based Encryption (ABE) to encrypt the keys of AES. However, there is an obvious problem in such a P2P approach. Data transfer can only be finished during two devices are online at the same time. Thus, Grouper uses multiple untrusted servers to synchronize user data rather than a P2P approach in Sweets.

6 Conclusion and Future Work

This paper describes Grouper, a framework using a Secret Sharing scheme and multiple untrusted servers, to develop mobile applications. Grouper provides two main functions: reliable data synchronization and group management. We implement Grouper's Web service in Java EE and clients in Objective-C. We use Grouper to develop applications including *Account Book*, *Notes* and *Test*. We evaluate Grouper from developer efforts and performance.

In the future, we will finish the application *Notes* on macOS, solve the synchronization order and JSON string redundancy caused by One-To-Many entity relationship, and improve the performance.

References

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

[2] Liao-Jun Pang and Yu-Min Wang. A new (t, n) multi-secret sharing scheme based on Shamir's Secret Sharing. *Applied Mathematics and Computation*, 167(2):840–848, 2005.

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

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

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

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

[7] 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.

[8] Rongchang Lai and Yasushi Shinjo. Sweets: A Decentralized Social Networking Service Application Using Data Synchronization on Mobile Devices. In *12th EAI International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2016.