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 requires central servers for storing shared data. The users of such mobile applications must fully trust the central server and their application providers. 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 services.

To address these problems, Grouper use multiple untrusted servers for data transfer. Data is divided into several pieces by the 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 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, this object in other devices can be 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 objects in these devices. We implement *Grouper Message* (Section 2.3) using a Secret Sharing (Section 2.3) scheme and untrusted servers (Section 2.2). Untrusted servers delete messages after a period of time, we call it interval time. *Grouper Message* implements reliable messaging over this feature (Section 2.3).

2.2 Threat Model

User data of mobile application using trusted central servers can be access by service provider easily and compromised by hackers. Therefore, we design Grouper based on data synchronization through multiple untrusted servers. Our untrusted servers have following features.

Firstly, servers performs device authentication. Servers generate access keys for group users. When a user want to synchronize data from untrusted servers, Grouper sends a HTTP request with access keys in request header.

Secondly, server keeps data temporarily. We define a period of time in which data can be kept in a server, we call it interval time. The data a sender uploaded to untrusted servers can exist within the interval time. After the interval time, it will be deleted.

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. 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 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.

To avoid threat in members inviting, group owners authenticate group members by a face-to-face way because inter-mobile device communication is secure. That requires 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 where n is three and k is two. 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

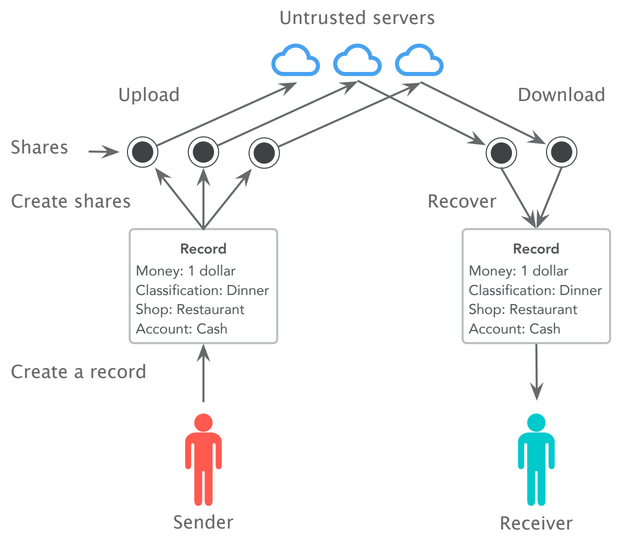

Figure 1: Data Transportation Flow

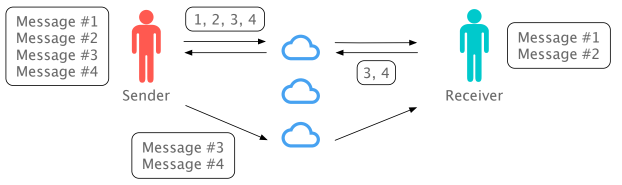

Figure 2: Reliable Synchronization

members in this group should synchronize this record, even if this record may be deleted by untrusted servers after a interval time. We call this problem reliable synchronization. A receiver can only download shares from untrusted server with the interval time. If he synchronize data after the interval time, he will miss the new record from the sender.

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 will check his local persistent store. In this situation, the receiver will find 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 *Grouper Message*. A Grouper message is a JSON string that contains the entities of the 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 entities of the application. We call them normal messages. Both confirm message and resend message contain control information about reliable multicast and need not resend. We call them control messages.

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

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

To confirm all group members have received all normal messages created by a user, the device of him need to send confirm message to group members periodically. In Grouper, when the application is launched, it should check the last send time of 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 should check the sequences. If some of them does not exist in its' persistent store, the device sends a resend message that contains sequence numbers that are not in the persistent store.

2.4 Group Management

2.4.1 Creating a Group

A group is created by its owner. 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 the group ID and 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 framework randomly when the application is launched at first the 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. Firstly, the new member sends user information and a node identifier to the owner. Owner saves user information and the node identifier of the new member to his device. Secondly, the owner register 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 existing members' list to the new member. After receiving them, the new member can access untrusted servers.

3 Implementation

Grouper consists of a client framework for developing iOS application and a Web service running on multiple un-

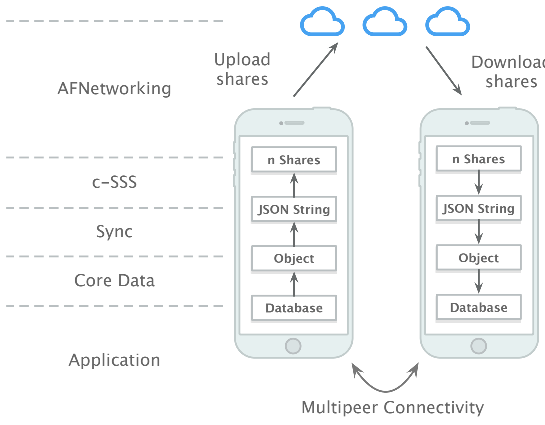

Figure 3: Architecture of Client

trusted 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.

Our web service provides RESTful API to transfer data with clients, and it keeps shares from devices temporarily. 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 entities. They are *Group*, *User* and *Transfer*. 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 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 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 cycle and object graph management, including persistence. Grouper uses it to manage the model layer objects.
- *Sync*[1], a modern JSON synchronization framework for *Core Data*. Grouper uses it to create JSON strings and parse JSON response and get it 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 revoke the RESTful API provided by our Web services.

3.3 Applications

Using Grouper framework, we are developing the following demo 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 notes for small group.

4 Evaluation

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

4.1 Developer Effort

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

```
// Initialize Grouper
grouper.setup(withAppId:"id",
  dataStack:DataStack(modelName: "name"))
// 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()
```

We have developed 2 applications including *Account Book* and *Test* with Grouper. As described in Table 1, based on the standard application without data synchronization, developers can developing the application with Grouper by adding little code.

4.2 Performance

The performance goal is to avoid significantly affecting the user experience with the application developed with

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

Table 1: Applications' lines of code

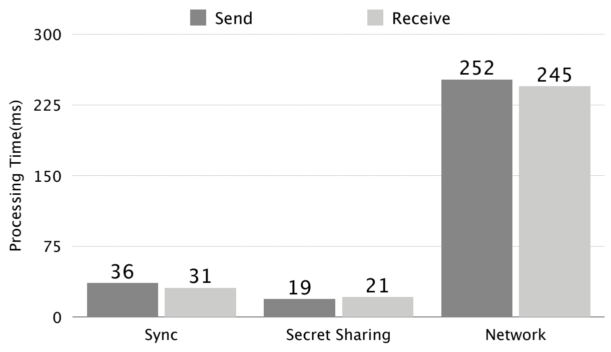

Figure 4: Sending and Receiving a Single Message

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 the wireless LAN network.

First, we measured processing times the benchmark app *Test* needs when a user creates objects during online. Figure 4 shows the processing time for sending and receiving a single message. Compared to network, data sync and secret sharing consume 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. Figure 5 shows the processing time for sending multiple messages. As the increasing of messages number, data sync and Secret sharing part increased linearly. The network part increased very slowly and sometimes decreased.

5 Related Work

To solve this problem in applications using central trusted servers, some applications use Peer to Peer (P2P) model 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 and sensitive 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, and users decrypt data by a browser extension. Like in Grouper, applications in Mylar can control how user data is shared and store data on multiple untrusted servers. Mylar builds its system on a browser with extension while Grouper uses mobile devices.

Sweets[8] is a decentralized social networking service(SNS) mobile application using data synchronization with P2P connections between devices. Sweets use 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

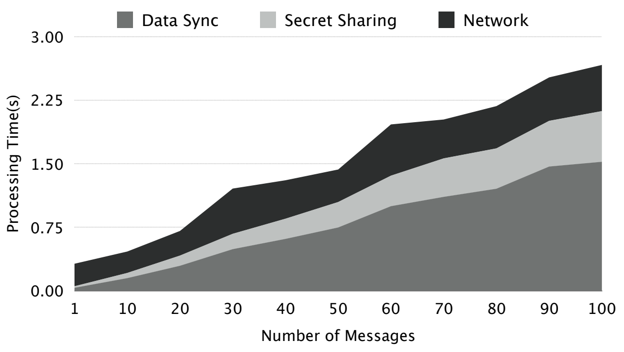

Figure 5: Sending Multiple Messages

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 P2P in Sweets.

6 Conclusion and Future Work

This paper describes Grouper, a framework using Secret sharing and multiple untrusted servers, to develop applications which is not relying on trusted central servers. Grouper provides two main functions including reliable data synchronization and group management. We implement the Web service in Java EE and the client in Objective-C. We use Grouper to develop applications including *Account Book*, *Notes* and *Test*. At last, we evaluate Grouper from developer effort 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 by retaining only 1 normal message for 1 object in untrusted servers.

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.