Mobile Cloud Computing: Implementation using Android and Firebase API

Balamurugan.V

Assistant Professor, Department of Computer Science and Engineering,
S.A. Engineering College, Chennai, Tamilnadu, India.

Abstract : Mobile computing is one of the wide spread research area in the domain of computer science as it attracts the researchers for its property of computing on-the-go. On the other hand, cloud computing deepens its application areas in almost all the sectors. Mobile cloud computing (MCC) gains its importance by facilitating the cloud services and computing facilities on the go. There have been a lot of survey and experimental frameworks for the implementation of MCC. In this paper, we provide a novel framework for developing an android application that provides data transmission between the mobile phones. Firebase cloud is integrated with the application for data storage. The aid of AJAX and JSON are yielded in order to provide efficient data transmission and consumption between the mobile nodes.

IndexTerms **- Androids, Mobile Computing, Cloud computing, Information security.**

I. INTRODUCTION

With the significant popularity of mobile terminals [1], people have begun to prefer to use mobile terminals to access and use the Internet over traditional terminals such as personal computers [2]. Mobile terminals have many advantages; for instance, such terminals are highly portable, fast, and interactive. Therefore, mobile terminals have become preferred for a large number of users, and application prospects are extremely broad. However, mobile terminals also have certain shortcomings. Of these shortcomings, the most serious is that their storage and computing capacities are limited [3]. Therefore, they cannot provide a large amount of storage or perform complicated calculations; they can only support certain lightweight file operations. Thus, mobile terminals prefer to export these storage-intensive and computationally complex tasks [4]. Cloud computing environments represent a good platform [5] and have substantial storage and computing resources; therefore, these tasks can be transferred to clouds [6]. Large files are stored in clouds.

We can download files from clouds when using mobile terminals; as a result, we effectively save substantial amounts of space [7]. We let Private Clouds (Prc) perform complicated calculations such as encryption [8]. In general, Prc can be built based on a trusted computing platform [9]. To ensure security, the private clouds cannot know the encryption key, that is, the remotely keyed encryption process. In practical applications, we often need to share data to the Public Clouds (Puc) for the sake of reducing the pressure on local storage and improving the convenience of using data.

Hence, the public clouds also need to store files. Thus, the private clouds share the cipher text file to the public clouds. Before a mobile terminal uses the file from the public clouds, the private clouds as a trusted third party [10] verify the data integrity to ensure security. Finally, the public clouds send the cipher text file back to the private clouds. The mobile terminal and private clouds perform remotely keyed decryption to allow the user to use the data. Through the above interaction in our proposed scheme, FREDP, the storage and computing overhead for the mobile terminal can be greatly reduced; we transfer large amounts storage and computing tasks to the private clouds. In addition, the security of the encryption key can also simultaneously be guaranteed for the remotely keyed encryption procedure. No parties other than the user can be allowed to know the encryption/decryption key and decrypt the file. Therefore, the confidentiality of the user's file can be ensured. The privacy of the user is perfectly assured in FREDP. In addition, by sharing the encrypted file from the private clouds to the public clouds, the sharing rate of the file can be greatly improved. In particular, the remotely keyed decryption method between the mobile terminal and the private clouds can guarantee the security of the decryption key as well as the file's usage by the user. Finally, the encryption/decryption performance under FREDP is not reduced.

II. RELATED WORK

Blaze M proposed the RKEP (Remotely Keyed Encryption Protocol) based on smartcards [1], where the complex calculations performed by a smartcard are transferred to the host, and the encryption key is retained by the smartcard. However, the computing ability of the host is limited, and it is not convenient to carry a smartcard for encryption and decryption process purposes. Oprea A presented a trusted third party PDA to protect the security of the user's key[2] such that the user can trade online securely.

However, the PDA also needs to be carried, reducing its usefulness in practice. In [3], the author proposed an energy-efficient integrity verification scheme for mobile clients to verify the integrity of the files stored on a cloud server using an incremental message authentication code[14]. This scheme does not transfer large numbers of verification jobs to the cloud server provider or the trusted third party to minimize the processing overhead on the mobile client. However, because the mobile terminal undertakes the intensive work of integrity verification, the mobile terminal can suffer from heavy burdens. Mobile terminals deliver the plaintexts to the public clouds directly; this method may threaten the security of the files.

In[5], a public provable data possession scheme was proposed for a resource-constrained mobile device that ensures privacy and confidentiality. In this scheme, a trusted third party is responsible for handling encoding/decoding, encryption/decryption, signature generation, and verification processes on behalf of the mobile user[4]. However, because mobile terminals perform the entire file encryption task and because the trusted third party knows the encryption key, the mobile terminals take on heavy workloads and the confidentiality of the encryption key is threatened. Encryption techniques such as type based proxy re-encryption and remotely keyed encryption technique are studied and they can be used in the proposed system. CloudExp framework is studied and helps in

implementing the cloud computing in mobile devices. Simulation of the MCC is surveyed because of the effects of implementing the MCC with respect to many QoS (Quality of Service) factors such as Data integrity, scalability, etc.

III. PROBLEM DEFINITION

A mobile computing model is in need that incorporates cloud computing services which is essential to provide feature-rich application to the users. The application should not violate security policies of cloud, should be scalable and flexible to the users need. Response time and energy consumption are two primary aspects for mobile systems that must be considered when making offloading decisions. Especially for resource-scarce devices, computation offloading is the key to empower these devices and augment their performance (not only energy), i.e., they can run code by means of the cloud that would never run locally.

IV. TECHNOLOGIES AND IMPLEMENTATION

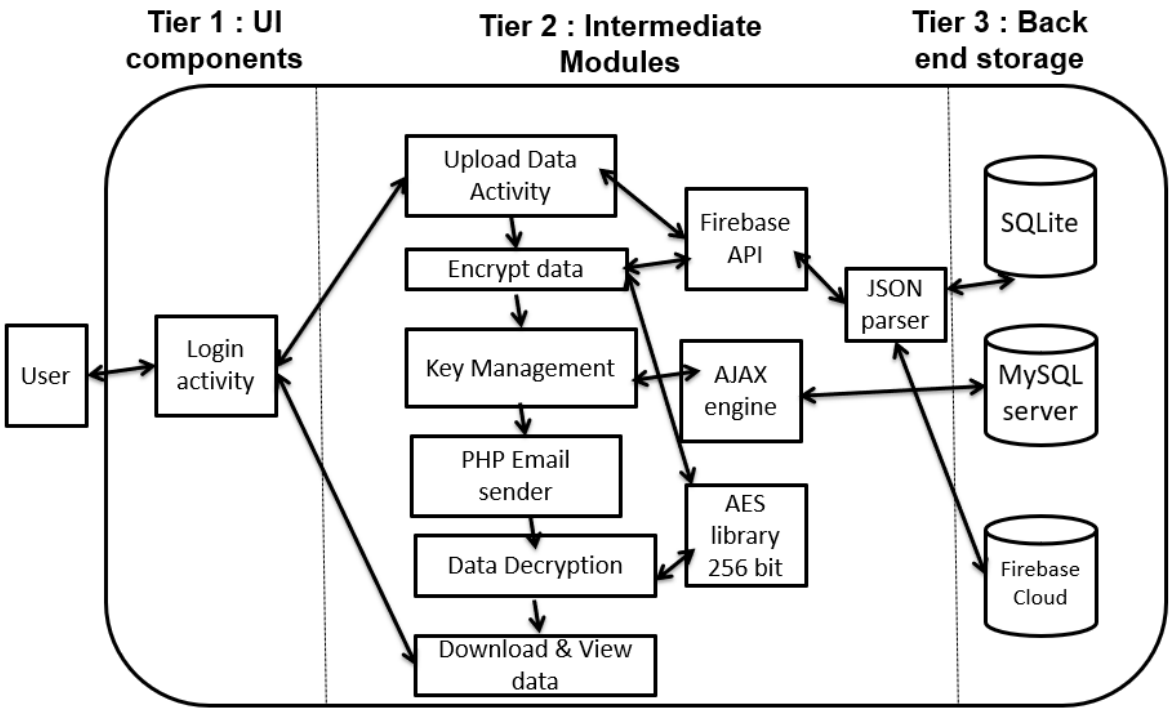

Figure 1. System Architecture

4.1 Data Encryption And Decryption

In MCC, local application modules in mobile terminals and remote computing modules in cloud data centers coordinate with each other to provide mobile users with a variety of on-demand services (such as VoIP, download services, etc.). The axioms for the data transmission under cryptographic terminologies are listed:-

- Message meaning rule logic axiom
  - This logic axiom means the following: If P believes that K is the shared key between P and Q and if P has received the result of X, which is encrypted by K, P believes that Q has sent X.
- For a public key, there is a similar axiom
- Temporary value verification rule logic axiom
  - This logic axiom means the following: If P believes that X is fresh and if P believes that Q has sent X, then P believes that Q believes X.

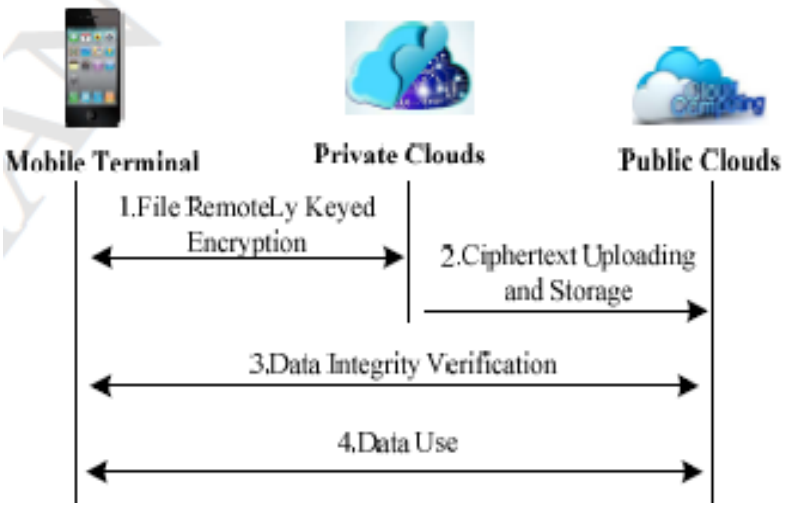

Figure 2. Data transmission in MCC

4.2 Firebase Cloud Integration

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably deliver messages at no cost. Using FCM, you can notify a client app that new email or other data is available to sync. Notification messages can be sent to drive user re-engagement and retention.

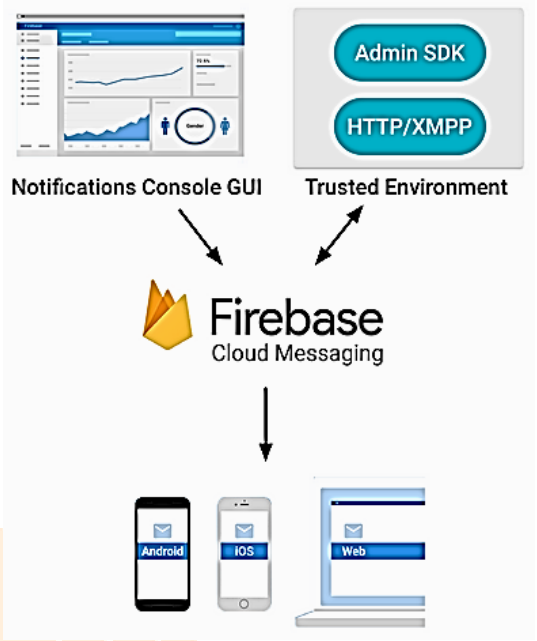

Figure 3. Firebase Cloud Messaging

In Android Studio, add the FCM dependency to your app-level build.gradle file:

```
dependencies {
    compile 'com.google.firebase:firebase-messaging:12.0.0'
}
```

A service that extends FirebaseMessagingService. This is required if you want to do any message handling beyond receiving notifications on apps in the background. To receive notifications in foregrounded apps, to receive data payload, to send upstream messages, and so on, you must extend this service.

```
<service ……
      <action android:name="com.google.firebase.MESSAGING_EVENT"/>
</service>
```

All Firebase Realtime Database data is stored as JSON objects. Using PUT, we can write a string, number, boolean, array or any JSON object to our Firebase database. In this case we'll pass it an object:

```
curl -X PUT -d '{
  "alanisawesome": {
    "name": "Alan Turing",
    "birthday": "June 23, 1912"
  }
}'
```

4.3 JSON

JSON stands for JavaScript Object Notation. It is an independent data exchange format and is the best alternative for XML. For parsing a JSON object, we will create an object of class JSONObject and specify a string containing JSON data to it. The last step is to parse the JSON. The Android platform includes the json.org library which allows processing and creating JSON files.

```
import org.json.JSONArray;
import org.json.JSONObject;
String jsonString = readJsonObjectFromSomeWhere();
try {
   JSONObject json = new JSONObject(jsonString);
   } catch (Exception e) {
      e.printStackTrace();
   }
public void writeJSON() {
   JSONObject object = new JSONObject();
   try {
      object.put("name", "Jack Hack");
      object.put("score", new Integer(200));
    } catch (JSONException e) {
      e.printStackTrace();
```

```
    }
    System.out.println(object);
}
```

4.4 ANDROID VOLLEY LIBRARY

Volley is an HTTP library that makes networking for Android apps easier and most importantly, faster. Volley is available on GitHub repository. Some salient features are: Automatic scheduling of network requests, Multiple concurrent network connections, Transparent disk and memory response caching with standard HTTP cache coherence, Support for request prioritization, Cancellation request API. You can cancel a single request, or you can set blocks or scopes of requests to cancel, Ease of customization, Strong ordering that makes it easy to correctly populate your UI with data fetched asynchronously from the network, Debugging and tracing tools.

Volley excels at RPC-type operations used to populate a UI, such as fetching a page of search results as structured data. It integrates easily with any protocol and comes out of the box with support for raw strings, images, and JSON. Volley is not suitable for large download or streaming operations, since Volley holds all responses in memory during parsing. For large download operations, consider using an alternative like DownloadManager. The core Volley library is developed on GitHub and contains the main request dispatch pipeline as well as a set of commonly applicable utilities, available in the Volley "toolbox." The easiest way to add Volley to your project is to add the following dependency to your app's build.gradle file:

```
dependencies {
    ...
    compile 'com.android.volley:volley:1.1.0'
}
```

When you fire a network request it is added to cache queue in the priority order. A cache dispatcher thread figures out whether it can service the request entirely from the cache or it needs to fire a network request for the response. In the case of a cache miss or if cached response has expired, the network dispatcher services the request, handles the parsing of the response and returns the response back to the main thread.

V. EXPERIMENTAL RESULTS AND FINDINGS

The proposed work is implemented in Android Studio 3.0.1 IDE with all latest API 27 and other features. A system that runs on Windows 10 with XAMPP server is deployed for the proposed system. When we conduct the experiment, we edit the codes to realize the remotely keyed encryption and decryption function in Java, and the interaction between the client and server is performed via socket communication.

For applications in mobile terminals, we usually focus on the operation time in the mobile terminal because the mobile terminal's resources are limited. Thus, we test the encryption and decryption performance. Finally, we compare the two types of times to check whether the FREDP can reduce the computing burden of the MT. Because the MT is a resource-constrained device and only has the capacity to perform lightweight operations, a file of 1 GB can be regarded as a large file. We test files of 1 KB, 10 KB, 100 KB, 1 MB, 10 MB, 100 MB and 1 GB in 5 respects: generating intermediate ciphertext time (GICT), generating encryption key time (GEKT), generating ciphertext time (GCT), generating decryption key time (GDKT) and generating plaintext time (GPT). Regardless of the encryption process and decryption process, the intermediate ciphertext is generated by the private clouds; thus, the time required to generate the intermediate ciphertext is only tested one time. The encryption and decryption performance parameters comparison are shown in Table1.

Table 1 Encryption Performance Comparison Table

File Size	GICT/ms	GEKT/ms	GDT/ms
20 KB	14	2	2
100KB	76	2	48
1MB	396	2	427
10MB	3277	2	3918
100MB	38400	2	46790
1GB	506678	2	658824

We test each size file three times when choosing the data and take the average to ensure that the data are more objective and accurate. From the table above, we can draw the conclusion that GICT, GDT and GPT are gradually increasing with increasing file size. However, GEKT and GDKT are mostly unchanged. In the remotely keyed encryption (decryption) process, the MT only encrypts (decrypts) the key; we can see that the time required is very short, only a few milliseconds. Such a lightweight computation is absolutely acceptable to the MT.

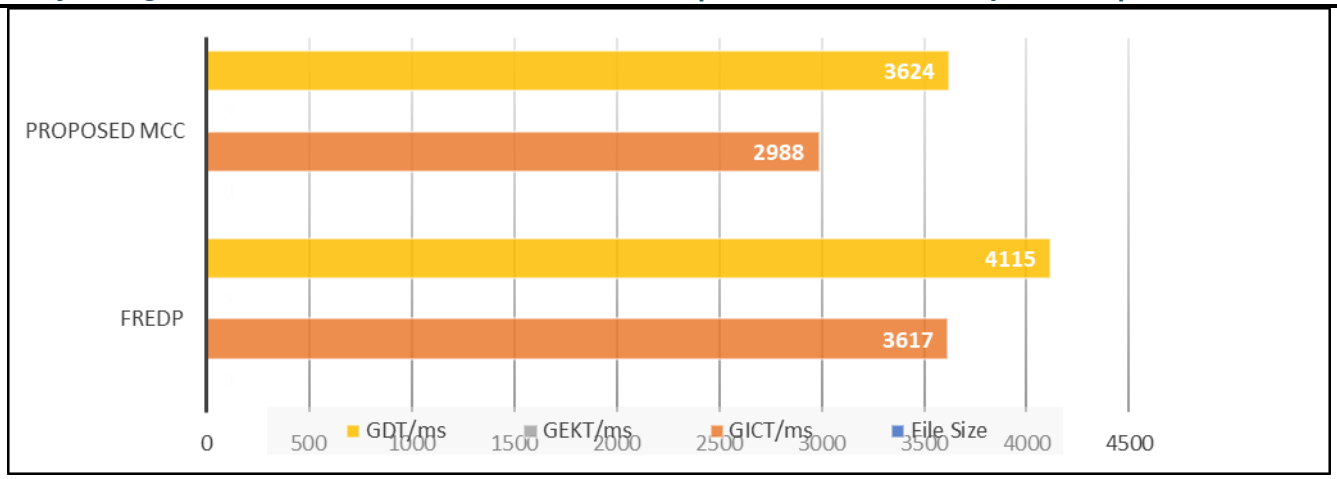

Figure 4. Comparison Chart

VI. CONCLUSION AND FUTURE ENHANCEMENT

An android application is built with Firebase integration for cloud storage purposes. The mobile devices will be able to transmit and receive data between them. Security, scalability and data integrity is focused. The project is completely FOSS (Free open Source Software) because all the technologies and tools used are FOSS. Therefore, the data uploading onto the cloud is enabled with security features. Multiple mobile devices are allowed.

For improved security, Elliptical Curve Cryptography (ECC) will be employed in this project work. Instead of XAMPP server running at a machine, a dedicated server with bandwidth of more than 100Mbps would handle further more requests from many simultaneous devices. All types of multimedia files such as an HD video file of high frame rate and quality should also be able to be loaded onto the cloud with required space. This project work could be implemented onto other platforms such as Windows, BB, iOS, etc.

REFERENCES

[1] Francisco Rodrigo Duro, Javier, Daniel, Oscar, Jesus, 'COSMIC: A hierarchical cloudlet based storage architecture for mobile clouds", Simulat. Modell. Pract. Theory (2014),http://dx.doi.org/10.1016/j.simpat.2014.07.007

[2] Huaming Wu, Energy-Efficient Decision Making for Mobile Cloud Offloading, IEEE TRANSACTIONS ON CLOUD COMPUTING, VOL. , NO. , 2017

[3] Huaming Wu, Multi-Objective Decision-Making for Mobile Cloud Offloading: A Survey, IEEE ACCESS, VOL. , NO. , 2018

[4] Jiang Zhang, Zhenfeng Zhang, and Hui Guo, November 2017, Towards Secure Data Distribution Systems in Mobile Cloud Computing , IEEE Transactions on Mobile Computing, Volume: 16, Issue: 11, Nov. 1 2017, DOI: 10.1109/TMC.2017.2687931

[5] Li Yang, Ziyi Han, Zhengan Huang, Jianfeng Ma, 23.12.2017, "A remotely keyed file encryption scheme under mobile cloud computing", *Journal of Network and Computer Applications* (2018), doi: 10.1016/j.jnca.2017.12.017

[6] Michael J. O'Sullivan, Integrating mobile and cloud resources management using the cloud personal assistant, Simulation Modelling Practice and Theory, Springer 2014.

[7] Muhannad Quwaider, Yaser Jararweh, Rehab, 2017, "Experimental framework for MCC system", First International Workshop on Mobile Cloud Computing Systems, Management, and Security (MCSMS-2015), Elsevier Science Direct, Pages 1147 – 1152

[8] Yun Li, Joint Optimization of Radio and Virtual Machine Resources with Uncertain User Demands in Mobile Cloud Computing, 1520-9210 (c) 2018 IEEE.