13 September, 2015

Secure Layer for Cloud Storage

Group name: Cloud Inspectors

Metehan Ozsoy, Jay P. Patel

1 Motivation

Storing files and sharing them securely on cloud storage has been a trending topic in the field of information security. There have been solutions provided by various applications that store encrypted user files together with the keys to decrypt them. As the cloud server may be an adversary, the files will be vulnerable since using the keys stored on the same cloud server could decrypt them. This violates the confidentiality. Our motivation in this work is to increase the confidentiality by preventing the server from accessing the decrypted files.

2 Problem Statement

Suppose we have a cloud that allows its users to store and share files among other users. How can we assure that unauthorized users and the cloud server itself do not have access to the content of the files?

3 Description of the Proposed Solution

We design an application for Android-based mobile devices that allows its users to store and share files with the other users as they would use current storage technologies like drop box or Google drive. But Our Application will act as security layer between those cloud storage and user. This layer will make sure no one except the user who upload file would be able to access the original content of file. We reach the level of confidentiality mentioned in the problem statement as follows:

Let u_1, u_2, ..., u_n be the users of the application.

Sign Up and Log In: Each user u_i creates a password p_i at the time of sign up where p_i is hashed by a hash function h(n). Also, u_i has a public key k_{pub}^i and a private key k_{pri}^i. We then store u_i, $h(p_i)$ and k_{pub}^i on the cloud server, and k_{pri}^i on u_i's mobile device. When the user u_i tries to log in, server checks if the entered username (u) and password (p) satisfies both $u = u_i$ and $h(p) = h(p_i)$.

Uploading: When the user u_i wants to upload a file f, a key k_f is generated that is uniquely associated with the file f. Then, k_f is used to encrypt f. The key k_f is sent to server that acts as secure layer, and the encrypted file is sent to Google Drive. Along with the k_f, checksum is also stored on secure server.

Downloading: User u_i downloads the encrypted file $Enc_{k_f}(f)$ from the Google server and the application gets the corresponding k_f from secure server and decrypts it on u_i's mobile device.

Sharing: Suppose u_i wants to share the file f with u_j. The application on u_i's mobile device gets u_j's public key k^j and uses it to encrypt k_f. The encrypted key file is sent to u_j's mobile device together with the link to $Enc_{k_f}(f)$. Then, u_j downloads $Enc_{k_f}(f)$ and decrypts it using k_f, hence has full access to the content of f. File sharing is completed without compromising confidentiality.

4 Details of Implementation

We will use Android Mobile Device as our application platform and mongo dB for storing all of the user's data, except files, and PHP scripting for processing data on secure server. Google drive API to store and retrieve files for each user. Application implementation will be done in Java language using Android Studio IDE. We will use Oracle Java Development Kit (JDK) 8. Last is github for source control.

5 Timeline and Work Distribution

- Creating the UI for the application for enlisting files (1 week, Jay and Metehan)
-- Establish a secure server (1 week, Jay)
-- Test the correctness of the encryption algorithms to be used in the application (1 week, Metehan)
- Mongo DB Structures for storing the checksums, keyfiles, user account info (1 weeks, Jay and Metehan)
- Implement Encryption algorithms (1 week, Jay and Metehan)
- Sign up and Log in procedure (1 week, Jay and Metehan)
- Uploading files to the Google drive (1 week, Jay and Metehan)
- Downloading files from the cloud server (1 week, Jay and Metehan)
- Sharing the files (2 weeks, Jay and Metehan)
- Testing (1 week, Jay and Metehan)