Secure Cloud Storage with Encrypted Data using File-Based Authentication

Jia Xu
National University of Singapore
xujia@comp.nus.edu.sg

Abstract

Cloud storage service is gaining popularity recently. Client-side deduplication is widely adopted by cloud storage services like Dropbox and MozyHome. Security flaws, which may lead to private data leakage, in the current client-side deduplication mechanism are found recently.

This paper presents a notion of File-based Authentication*, that is, a user authenticates himself/herself using the file he/she possesses as the secret information. File-based Authentication can be applied to protect confidentiality of users' sensitive data file in the cloud storage from both the third party attackers and the semi-honest (honest but curious) cloud server itself. The proposed solution enables efficient client-side deduplication and sharing of encrypted data in the cloud storage, where the encryption key is chosen by a user* independently[1] *and is kept secret from the cloud storage server.*

Furthermore, the surprising part is that, a secure cloud storage system without registration for users can be constructed based on the proposed solution: All owners of a file F share an a priori account with hash value $h(F)$ as identity and sensitive file F as password, without knowledge of each other and without registration on the cloud storage server, since the first uploading of F to the cloud will function as the registration of account $(h(F), F)$ in the cloud storage server.

Keywords

Privacy, Client-side deduplication, Proof of Ownership, Client-side Sharing, File-based Authentication, Registration-free Cloud Service

1. Introduction

Cloud storage service is gaining popularity. To reduce resource consumption, several cloud storage services including Dropbox [Dro] employs client-side deduplication. That is, when a user tries to upload a file to the server, the server checks whether this particular file is already in the cloud and saves the uploading process if it is. In this way, every single file will have only one copy in the cloud. Recently, Halevi *et al.* [SH11] presented an attack on current client-side mechanism in popular cloud storage service like Dropbox: If the adversary *somehow* has the hash value of a file stored in the cloud storage, he could fool the cloud server that he has the file by presenting only the hash value in the client-side deduplication process, and thus gain access to that file in the cloud.

Halevi *et al.* [SH11] proposed a primitive called *Proofs of Ownership* to allow an owner of a file to prove to the server that he/she indeed has that file in an efficient way. Informally, in a Proofs of Ownership scheme, if an adversary *somehow* has some limited knowledge on a file, he cannot gain significant more knowledge of that file via the client-side deduplication process with the cloud storage server. Shortly, information leakage cannot be amplified in the deduplication process.

Proofs of Ownership alone achieves strong security. However, when deployed in a cloud storage service, the security of the overall system is limited by other components, like user authentication system, of the cloud service. In a practical cloud storage service secured by PoW [SH11], adversary still can obtain the large data file from a short secret value—the user name and password of (one of) the owner of that file. Furthermore, the root account password of the cloud server can be considered as a short representation of all files stored in the cloud storage. Once an adversary *somehow* obtains the root password, he can access every single file in the cloud, which is a huge information leakage amplification.

Furthermore, all Proofs of Ownership schemes proposed in Halevi *et al.* [SH11] have to trust the cloud server and only protect the data confidentiality from third party attackers, since these schemes cannot support encrypted data file. However, sensitive data files are likely to be encrypted on the client side by users before uploaded to the cloud storage.

Different from PoW, this paper deals with encrypted data files, and takes the whole cloud storage system into account, and aims to protect confidentiality of users' data in the cloud storage against both the third party attackers, and the semi-honest cloud server itself, who is curious about the content of users' sensitive data but will guarantee the integrity of users' data. If the cloud server is semi-honest, our solution achieves higher level of security, compared to cloud storage system secured with PoW; if the cloud server is honest, our solution achieves the same level of security.

It is known that cloud storage service SpiderOak [Spi] provides data deduplication over encrypted data in the cloud. However, in SpiderOak service, users' data files are encrypted on the client side by SpiderOak client software which is closed source, and apparently the encryption key is chosen by the SpiderOak client software. Instead, in this paper, a user's data file is encrypted using a public encryption algorithm with the encryption key chosen by the user independently, without any negotiation with other owners of the same file or the cloud server. We emphasize that the encryption key should be kept safely from the cloud server (including the client software of the cloud service).

1. Without negotiation with other owners of the same data file.

1.1. Our results

Our results and contributions in this paper are summarized as below.

1.1.1. File-based Authentication. This paper presents a notion of *File-based Authentication*. In a File-based Authentication system, an owner of a file F can authenticate himself/herself to the cloud server, using the hash value $h(F)$ as identity and the file F as the password, with a zero-knowledge proof.

1.1.2. Client-side Deduplication. To apply File-based Authentication in cloud storage, each data file will be encrypted using a deterministic encryption scheme using a one-time key and this one-time key will be encrypted using the file F as secret key. Next, the resulting ciphertexts of the data file and the one-time key will be uploaded to the cloud storage. An owner of file F' cannot have access to the content of the file F stored (in ciphertext) in the cloud storage, unless he/she can authenticate to the cloud server with identity $h(F)$. A secure File-based Authentication ensures that this authentication will succeed only if $F' = F$, except a negligible probability of exceptions.

1.1.3. Client-side Sharing. Current sharing mechanism in cloud storage is implemented on the server side: If a user Alice grants another user Carol to access Alice's file F, the cloud storage server Bob will provide a soft link of F to the user who can authenticate himself/herself with Carol's account id. With all file encrypted before uploaded to the cloud, the server itself is not able to access the content of users' files, and cannot grant any other users to access a particular file, neither. Therefore, the server-side sharing mechanism does not work with encrypted data.

Our solution supports efficient client-side sharing mechanism: To share file F with Carol, Alice just passes the corresponding secret key sk_F to Carol in a secure channel. With the secret key sk_F as trapdoor, Carol can convince Bob to believe that she has F in the client-side deduplication process, and then can download the ciphertext of F from the cloud storage Bob and decrypt it using sk_F. We emphasize that different files have independent secret keys.

1.1.4. Secure Cloud Storage Service without Registration. The File-based Authentication allows a cloud storage serice without registration: The first uploading of a file F to the cloud functions as registration of the user account $(id = h(F), pin = F)$ to the cloud server automatically, and any owner of the same file F can authenticate to the server with $h(F)$ as id and F as pin. The sensitive file F is the a priori shared secret among all owners of F. Registration-free cloud service together with anonymous network routing make it much more difficult for the cloud server to track users' profile (like the user-file ownership graph) and usage history.

2. Formulation

2.1. Background on Proofs of Ownership

Halevi *et al.* [SH11] proposed Proofs of Ownership schemes to resolve the security flaw in the client-side deduplication mechanism implemented in several cloud storage services, including Dropbox. However, Proofs of Ownership schemes [SH11] has several limitations.

First, PoW schemes [SH11] cannot be applied to encrypted data and so cannot prevent the cloud storage server from accessing the user data stored in the cloud. Second, although PoW alone achieves strong security, a cloud storage server integrated with PoW schemes still suffers from data-leakage-amplification. For example, the user account (i.e. user id and password) of (one of) the owner of a data file can be considered as a "short" representation of that file. More worsely, the root account of the cloud storage system can be viewed as a "short" representation of all data files stored in the cloud storage.

2.2. File-based Authentication

A $\mathcal{FBA}$ scheme $(\mathcal{E}, \mathcal{D}, \langle \mathcal{P}, \mathcal{V} \rangle)$ consists of two PPT algorithms $\mathcal{E}$ and $\mathcal{D}$ and a pair of PPT interactive algorithms $\langle \mathcal{P}, \mathcal{V} \rangle$:

- $\mathcal{E}(F, 1^\lambda) \to (sk, C, t)$: The probabilistic encoding algorithm $\mathcal{E}$ takes as input a data file F and a security parameter λ, and outputs a private key sk, a ciphertext C and an authentication tag t.
- $\mathcal{D}(sk, C) \to F$: The deterministic decoding algorithm takes as input a private key sk and a ciphertext C, and outputs a file F.
- $\langle \mathcal{P}(F), \mathcal{V}(t) \rangle \to (sk; \quad b)$: The prover algorithm $\mathcal{P}$ with a file F as input interacts with the verifier algorithm $\mathcal{V}$ with an authentication tag t as input. At the end of interaction, the prover algorithm $\mathcal{P}$ gets output sk and the verifier algorithm $\mathcal{V}$ gets output $b \in \{\texttt{Accept}, \texttt{Reject}\}$.

Definition 1 (Correctness): We say a $\mathcal{FBA}$ system $(\mathcal{E}, \mathcal{D}, \langle \mathcal{P}, \mathcal{V} \rangle)$ is *correct*, if the following conditions hold: For any data file F and any positive integer λ, and $(sk, C, t) := \mathcal{E}(F, 1^\lambda)$,

- $\mathcal{D}(sk, C) = F$.
- $\langle \mathcal{P}(F), \mathcal{V}(t) \rangle = (sk, \texttt{Accept})$.

Before defining the security requirement for a $\mathcal{FBA}$, we generalize the notion of semantic-secure. Let us define a (δ, ϵ)-Semantic-Secure game $\mathsf{Game}_{\delta,\epsilon}$ for an encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as below.

Commit. The adversary $\mathcal{A}$ chooses a set $\mathbb{M}$ of equal length bit-strings, and a distribution Π over the set $\mathbb{M}$, such that the min-entropy of distribution Π is at least ϵ. The adversary $\mathcal{A}$ gives $(\mathbb{M}, \Pi)$ to the challenger $\mathcal{C}$.

Challenge. The challenger $\mathcal{C}$ samples a message M from the set $\mathbb{M}$ under distribution Π. Next, the challenger randomly chooses δ number of encryption keys $k_i := \mathsf{KeyGen}(1^\lambda), i \in [\delta]$, and encrypts the message M under each key k_i to obtain a ciphertext $C_i := \mathsf{Enc}(k_i, M)$. All δ ciphertexts $(C_1, \ldots, C_\delta)$ are given to the adversary $\mathcal{A}$.

Guess. The adversary $\mathcal{A}$ outputs a guess $M' \in \mathbb{M}$, and wins this game if and only if $M' = M$.

Definition 2: We say an (symmetric) encryption scheme $(\mathsf{KeyGen}, \mathsf{E}, \mathsf{D})$ is (δ, ϵ)-Semantic-Secure under Cross-Key Ciphertext-Only Attack, if for any PPT adversary $\mathcal{A}$, the probability

$$\mathsf{Pr}[\mathcal{A} \text{ wins } \mathsf{Game}_{\delta,\epsilon}] \leq 2^{-\epsilon} + negl.$$

Note that this (δ, ϵ)-Semantic-Security under Cross-Key Ciphertext-Only Attack can be easily adapted for public key encryption scheme.

Roughly, if an encryption scheme is (δ, ϵ)-Semantic-Secure under Cross-Key Ciphertext-Only Attack, and a plaintext M has min-entropy at least ϵ, given δ number of encryptions of plaintext M under δ random encryption keys respectively, no efficient adversary can find more than $\epsilon - 1$ bits of the plaintext M. The standard semantic security can be considered as an extreme (strong) case: $(1, 1)$-Semantic-Secure under Cross-Key Ciphertext-Only Attack.

Proxy re-encryption scheme might achieve such (δ, ϵ)-semantic-security. Several negative results [MR08] on the security of RSA scheme under this attack model are known.

Definition 3 (Privacy-Preserving): We say a $\mathcal{FBA}$ system $(\mathcal{E}, \mathcal{D}, \langle \mathcal{P}, \mathcal{V} \rangle)$ is (δ, ϵ)-*Privacy-Preserving*, if the following conditions hold:

- $(\mathcal{E}, \mathcal{D})$ forms a (δ, ϵ)-Semantic-Secure encryption scheme under *Cross-Key Ciphertext-Only Attack*.
- $\langle \mathcal{P}(F'), \mathcal{V}(t) \rangle$ is **ZKP** of the statement $F' = F$, where $(sk, C, t) := \mathcal{E}(F, 1^\lambda)$.

We require the system to be secure, even if multiple interactive proof sessions (multiple provers interacts with a single or multiple verifiers) run simultaneously.

3. File-based Authentication Scheme

3.1. Overview

The idea is that: The sensitive data file is encrypted using a deterministic encryption scheme. We allow a client to access the private encryption key if and only if the client indeed has access to the exactly same file. Informally, we encrypt secret information using the data file as the private encryption key.

3.2. Construction

Let $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a digital signature scheme, and $\mathsf{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a deterministic symmetric encryption scheme (e.g. AES). Let h be a full domain secure hash function (e.g SHA256) and $\{\mathcal{H}_s\}$ be a keyed-hash family (both h and $\mathcal{H}_s$ will be treated as random oracles). Let $\oplus$ denote the XOR operator between two equal length bit-strings. We construct a $\mathcal{FBA}$ scheme $(\mathcal{E}, \mathcal{D}, \langle \mathcal{P}, \mathcal{V} \rangle)$ as below.

$\mathcal{E}(F, 1^\lambda)$.

1) $k_1 := \mathsf{E.KeyGen}(1^\lambda)$; $k_2 := \mathsf{E.KeyGen}(1^\lambda)$
2) $(spk, ssk) := \mathsf{S.KeyGen}(1^\lambda)$
3) $s \xleftarrow{\$} \{0,1\}^\lambda$
4) $sk = (k_1, k_2, ssk)$
5) $C := \mathsf{E}_{k_1}(F)$
6) $t := (\mathsf{E}_{k_2}(ssk), \mathcal{H}_s(F) \oplus (k_1 \| k_2), s, spk)$
7) Output (sk, C, t).

$\mathcal{D}(sk, C)$.

1) Parse sk as $(\hat{k}_1, \hat{k}_2, \hat{ssk})$
2) Parse C as $\mathsf{E}_{k_1}(F)$
3) $F' := \mathsf{E.Dec}_{\hat{k}_1}(\mathsf{E}_{k_1}(F))$
4) Output F'.

$\langle \mathcal{P}(F'), \mathcal{V}(t) \rangle$.

V1: Parse t as $(\mathsf{E}_{k_2}(ssk), \mathcal{H}_s(F) \oplus (k_1 \| k_2), s, spk)$. Choose a random message M. Send $(M, \mathsf{E}_{k_2}(ssk), \mathcal{H}_s(F) \oplus (k_1 \| k_2), s)$ to the prover.

P1: Compute the private key sk' and the signature σ for message M as below.

a) $k_1' \| k_2' := \mathcal{H}_s(F') \oplus (\mathcal{H}_s(F) \oplus (k_1 \| k_2))$
b) $ssk' := \mathsf{E.Dec}_{k_2'}(\mathsf{E}_{k_2}(ssk))$
c) $sk' := (k_1', k_2', ssk')$
d) $\sigma := \mathsf{S.Sign}_{ssk'}(M)$.

Keep the private key sk' safely and send the signature σ to the verifier.

V2: If $\mathsf{S.Verify}_{spk}(M, \sigma) = \texttt{Accept}$, then output `Accept`; otherwise output `Reject`.

Theorem .1: Suppose E is a (δ, ϵ)-Semantic-Secure under Cross-Key Ciphertext-only Attack and hash function h and $\mathcal{H}_s$ are random oracles. Then the above construction $(\mathcal{E}, \mathcal{D}, \langle \mathcal{P}, \mathcal{V} \rangle)$ is correct and (δ, ϵ)-Privacy-Preserving.

3.3. A Concrete Instantiation

We may choose AES-256 (i.e. AES with 256 bits key) as the underlying encryption scheme E, DSA or Elliptic-Curve DSA as the underlying digital signature scheme S, and the hash function SHA256 as the full domain hash function h.

We may construct the universal hash family $\{\mathcal{H}_s : \{0,1\}^* \to \{0,1\}^{512}\}$ in the following way:

1) Without loss of generality, assume the bit length of a hash input x is 512ℓ for some integer ℓ. Treat the input x as an ℓ-dimensional vector $\vec{\boldsymbol{x}}$ from the space $\left(\{0,1\}^{512}\right)^{\ell}$.
2) Generate a vector $\vec{\boldsymbol{s}}$ from s using a pseudorandom function PRF: $\vec{\boldsymbol{s}} := (\mathsf{PRF}_s(1), \ldots, \mathsf{PRF}_s(\ell)) \in \left(\{0,1\}^{512}\right)^{\ell}$.
3) Output the hash of the inner product $\langle \vec{\boldsymbol{x}}, \vec{\boldsymbol{s}} \rangle \mod 512$ as the hash value of x: $\mathbf{SHA256}(\langle \vec{\boldsymbol{x}}, \vec{\boldsymbol{s}} \rangle \mod 512)$.

4. Application of File-based Authentication in Cloud Storage Service

Let $(\mathcal{E}, \mathcal{D}, \langle \mathcal{P}, \mathcal{V} \rangle)$ be a correct and privacy-preserving File-based Authentication scheme. We can apply this scheme to achieve secure client-side deduplication and sharing of encrypted data in cloud storage service. Furthermore, a registration-free secure cloud service can be constructed.

4.1. Client-side Deduplication

4.1.1. First Upload of file F. Suppose Alice is the first user who uploads data file F to the cloud storage server Bob. Alice chooses a security parameter λ and runs the encoding algorithm $\mathcal{E}$: $(sk, C, t) := \mathcal{E}(F, 1^{\lambda})$. Then Alice uploads the ciphertext C, authentication tag t and the hash value $h(F)$ to Bob, and keeps the private key sk safely. This process can be viewed as enrollment of account $(h(F), F)$ to the cloud server.

4.1.2. Subsequent Upload of file F. Suppose Carol is one of the subsequent users who try to upload the file F' to Bob. Carol sends the hash value $h(F')$ to Bob. If Bob finds a matching hash value in his cloud storage, then a duplication event occurs and Bob finds the corresponding authentication tag t. Next, Carol,who is running the prover algorithm $\mathcal{P}(F')$, interacts with Bob, who is running the verifier algorithm $\mathcal{V}(t)$. At the end of interaction, Carol obtains a private key sk' and Bob obtains $b \in (\texttt{Accept}, \texttt{Reject})$.

The correctness of our $\mathcal{FBA}$ scheme guarantees that if $F' = F$, then $sk' = sk$ and $b = \texttt{Accept}$. The security of our $\mathcal{FBA}$ scheme guarantees that if the min-entropy of file F to Carol is at least ϵ, then with overwhelming high probability that $sk' \neq sk$ and $b = \texttt{Reject}$.

4.2. Client-side Sharing

Since the cloud storage server has no access to Alice's encrypted data file, the straightforward server-side sharing mechanism, which provides a soft link of the data file to Carol if Alice authorizes Carol to access a particular file, does not work on encrypted data: This mechanism only allows Carol to download the encrypted data with no means to decrypt it.

When all data files are encrypted, our scheme allows client-side sharing between friends. To share a file F with Carol, Alice can just pass the corresponding secret key sk_F to Carol via a secure channel. With the secret key sk_F as trapdoor, Carol can convince the cloud storage server Bob that she *has* file F, and thus can download the ciphertext of F from Bob and decrypt it using sk_F. We emphasize that in our scheme, different files will have independent secret keys.

4.3. Cloud Storage Service without Registration

We can view our solution in this way: all owners of a file F show the same account $(id = h(F), pin = F)$ and the first uploading of F and the registration of account $(h(F), F)$ are in a single step. From the users' point of view, there is no additional registration process before using the cloud service, since the first use (w.r.t. a file F) of cloud service is considered as registration by the server. Compared with the cloud storage service with user registration, our solution together with anonymous network routing protects user privacy better, in the sense that it will be more difficult for the cloud server to track users' profiles and usage history.

5. Conclusion

In this paper, we proposed a notion of "File-based Authentication", i.e. the owner of a file can authenticate himself/herself to a cloud server with the file as secret authentication key. We gave an efficient construction of File-based Authentication, and applied to support secure client-side deduplication and sharing of encrypted data in cloud storage service. Furthermore, File-based Authentication implies that a secure cloud storage service without registration of user accounts can be constructed. Building a secure cloud storage service over P2P network by integrating our solution with Proofs of Retrievability is in the future work.

References

[Dro] Dropbox. Dropbox. http://www.dropbox.com/.

[MR08] Alexander May and Maike Ritzenhofen. Solving systems of modular equations in one variable: how many RSA-encrypted messages does eve need to know? In *PKC '08*, pages 37–46, 2008.

[SH11] Benny Pinkas Alexandra Shulman-Peleg Shai Halevi, Danny Harnik. Proofs of ownership in remote storage systems (to appear in ccs '11). Cryptology ePrint Archive, Report 2011/207, 2011. http://eprint.iacr.org/.

[Spi] SpiderOak. Spideroak. https://spideroak.com/.