Future Generation Computer Systems 62 (2016) 119–127

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Highly privacy-protecting data sharing in a tree structure

S. Canard [a,*], J. Devigne [b]

[a] Orange Labs, Applied Crypto Group, Caen, France
[b] DGA Maîtrise de l'information, Bruz, France

HIGHLIGHTS

- We propose a way to store data in a non trusted cloud storage provider.
- We make use of the advanced cryptographic tool called "proxy re-encryption".
- We provide a way for customers to share files and folder by modifying the standard proxy re-encryption paradigm.
- We show that a true implementation of such system is very efficient and can easily be embedded into commercial products.

ARTICLE INFO

Article history:
Received 1 May 2015
Received in revised form
14 November 2015
Accepted 27 January 2016
Available online 2 March 2016

Keywords:
Cloud storage
Confidentiality
Privacy
Cryptography

ABSTRACT

In this paper, we investigate the way to efficiently implement a highly privacy-protecting data sharing system in a cloud storage context. We suppose that several customers want to share some sensitive and personal data that are stored on a non-trusted cloud storage system, in such a way that the latter has no way to obtain the data in clear. For this purpose, we make use of an advanced cryptographic tool called a "proxy re-encryption" scheme. In this context, our contribution is twofold. We first modify existing proxy re-encryption schemes in such a way that customers can now manage dynamically a tree structure for their shared document, which was not possible with existing systems. We then present the first true implementation of such system where each client makes use of a smartphone to upload, download and share his/her documents. This way, we show that such system is really practical for a real-life use.

1. Introduction

Network data storage is a widespread option and it is today easy to find such solutions for any digital content. A key issue is the confidentiality of stored data, which becomes even trickier when the owner wants to share her/his data. It is then crucial to offer solutions that combine storage, sharing and confidentiality of data, without changing the user experience.

Cloud storage is a system by which users can store their personal and sensitive data in a "secure way", such that they are later able to access their data anywhere, at any time, from any authorized devices. The most interesting case is when the cloud storage is dynamic since it permits to control the access to the data by adding/deleting new devices and users. The idea behind cloud storage is that data are stored as if they were in a safe, where the cloud storage plays the role of an access control to this safe. In reality, the data are encrypted but, most of the time, the cloud server has the decryption key and manages the rights for each user to access or not the data.

This is a critical problem in the case of private sensitive data such as administrative documents (*e.g.* identity cards, bills or pay sheets) or, more generally, personal data. This is also a tricky point in the case of confidential documents owned by a business enterprise and shared between collaborators or with trading partners. In fact, this problem can be easily solved by simply encrypting the data before sending it to the cloud/safe. However, this only provides a backup service, and not a secure storage one with practical features. More precisely, it does not work when one wants to share the stored data with other customers, or between his own devices.

One solution is to share the decryption key but this is naturally a "bad" solution, especially regarding security. The compromise of one device (lost or stolen) necessitates all remaining devices to update their keys and all the stored documents should be re-encrypted again, with a new generated key.

Another solution consists in duplicating the data for all authorized entities or, at most, the secret key used to encrypt

* Corresponding author.
E-mail addresses: sebastien.canard@orange.com (S. Canard), julien.devigne@gmail.com (J. Devigne).

http://dx.doi.org/10.1016/j.future.2016.01.019
0167-739X/

the large file (in case of an encapsulation mechanism). Regarding security, this solution is much better since the compromising of one device only necessitates the deletion of the related encrypted documents, but does not impact other devices/entities. The main issue regarding this solution is its flexibility. More precisely, if one wants to withdraw the "read" right to all his/her documents, the server has to delete all corresponding entries. And if one wants to later share a document with someone else, he/she has to be online to create the new version of the encrypted secret key. New solutions have recently emerged by using advanced cryptography which allows to combine privacy and functionalities, and seem to be the best compromise between security, efficiency and usability.

One can find different ways to solve the above problem in the cryptographic literature. Some papers make use of attribute based encryption [1,2], with the particularity that it can easily manage hierarchical structures. It seems also possible to use broadcast encryption [3] to provide a way to share an encrypted document with a group of users. But some others prefer to use proxy re-encryption [4,5] since it permits to easily manage simple sharing between customers. In this paper, we focus on the latter since, from our point of view, it is best suited for a peer-to-peer sharing in non hierarchical structure, on which we have worked.

PROXY RE-ENCRYPTION FOR SECURE STORAGE. In [5], Ateniese et al. have proposed a privacy-preserving architecture for distributed storage which makes use of a proxy re-encryption (PRE) scheme [4]. In a nutshell, a PRE allows a user to delegate its decryption capability in case of unavailability. To do so, this user, Alice, computes a *re-encryption key* $\mathsf{rk}_{A\to B}$ which is given to a proxy. The key $\mathsf{rk}_{A\to B}$ allows the proxy to transform a ciphertext intended to Alice into one intended to Bob. While doing this, the proxy cannot learn *any* information on the plaintexts nor any secret key.

A similar PRE based storage system has also been proposed in the case of cloud storage in [6]. With such a system, where the cloud plays the role of the proxy, the access to a plaintext is only permitted to authorized users, while the cloud cannot derive the plaintext from the stored ciphertext. A data can *e.g.* be stored on a dedicated cloud storage using Alice's public key. If Bob can access this document, the proxy/cloud makes use of the re-encryption key from Alice to Bob. Similarly, if Alice owns several devices, one document encrypted with the key on one of them can be re-encrypted for another one. This is done without needing them to share the same secret decryption key, and in a private way since the storage provider does not obtain the data in plain.

The (cloud) storage system of Ateniese et al. [5] makes use of a (1) *unidirectional* and (2) *single-hop* scheme, which means (1) that with a re-encryption key $\mathsf{rk}_{A\to B}$, a proxy cannot translate Bob's ciphertexts into ciphertexts intended to Alice and (2) that once a message has been moved into a ciphertext intended to Bob, no more transformation on the new ciphertext intended to Bob is possible. It also exists in the literature several PRE schemes which are *bidirectional*, meaning that they allow a symmetrical transformation, and *multi-hop*, meaning that several "consecutive" translations of ciphertexts are possible. The way to manage at the same time multiple users and multiple devices in such system has recently been proposed by Canard and Devigne [7]. They thus design the concept of combined proxy re-encryption where one can manage at the same time unidirectional and single-hop PRE (to share documents between users) and bidirectional and multi-hop PRE (to share document between several devices of a single user).

OUR APPROACH. In this paper, we focus on the fine-grain management of the rights. A standard PRE scheme has an "all or nothing" sharing property. If the re-encryption key is generated by Alice, then the proxy can re-encrypt for Bob any document initially encrypted to Alice. There is no way for Alice to restrict what the proxy can re-encrypt or not, except by trusting it. But if the storage space is structured as a tree (as shown in Fig. 1), Alice may want to only share a specific folder, or a specific files, but not all her files.

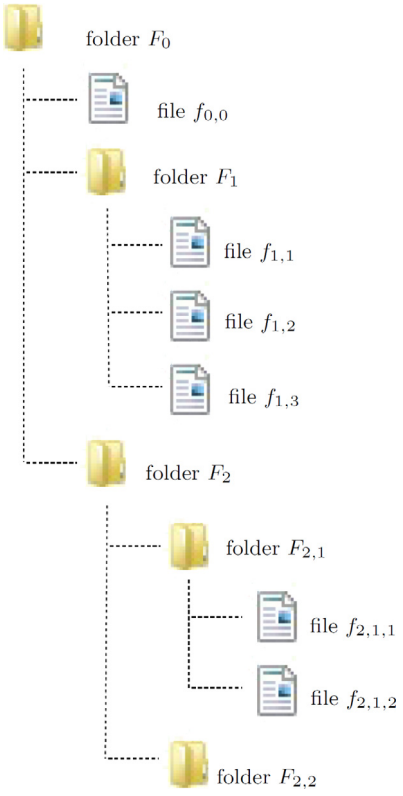

Fig. 1. A tree structure.

For this purpose, it exists *conditional* PRE. In such variant, the encryption process is related to a chosen condition c_1, and the re-encryption key is generated under a condition c_2. Then, if $c_1 = c_2$, the re-encryption process outputs a value that can be normally decrypted, whereas if $c_1 \neq c_2$ then the decryption process outputs an error message.

But this is not enough in our case. In fact, this can only be helpful to manage a file by file sharing, but not a true hierarchy, since there is no possible link between a file and its mother's folder. We need more work that we have done in this paper by designing a two directions conditional PRE, as illustrated in Fig. 2.

We then attach to each uploaded file a unique condition, defined during the encryption process, and denoted $\gamma_{f_{2,1,1}}$ for file $f_{2,1,1}$ for example. We then obtain the ciphertext $\mathsf{Enc}(\mathsf{pk}_A, f_{2,1,1}, \gamma_{f_{2,1,1}})$, using Alice's public key pk_A. If Alice wants to *e.g.* give the rights to Bob for folder F_2, she computes a re-encryption key, from Alice to Bob, under a condition related to F_2. Such re-encryption key is denoted $\mathsf{rk}_{A\to B,F_2}$ and is sent to the proxy. This re-encryption permits a vertical transformation between users, if and only if the conditions match. We then add an horizontal transformation inside the tree, using additional re-encryption keys such as $\mathsf{rk}_{f_{2,1,1}\to F_{2,1},A}$ or $\mathsf{rk}_{F_{2,1}\to F_2,A}$. These keys permit to go back up the tree from a file to the root folder by modifying the condition attached to the ciphertext.

More precisely, for each couple (file, folder) or (folder, folder) in the path from the file to the root, Alice needs to compute a re-encryption key (but only once for each link between folders). For example, there is a re-encryption key $\mathsf{rk}_{F_{2,1}\to F_2,A}$ which permits to modify the condition related to the ciphertext (to which this re-encryption is used), without modifying the underlying public key. The whole result, for our example, is given in Fig. 2.

Finally, when the encrypted file is related to a condition for which it exists a re-encrypted key from Alice to Bob (in our