



Universidad  
Carlos III de Madrid



This is a postprint version of the following published document:

González-Manzano L., de Fuentes J.M., Choo K.K.R. (2017) ase-PoW: A Proof of Ownership Mechanism for Cloud Deduplication in Hierarchical Environments. In: Deng R., Weng J., Ren K., Yegneswaran V. (eds) Security and Privacy in Communication Networks. SecureComm 2016. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 198. Springer, Cham  
DOI: [https://doi.org/10.1007/978-3-319-59608-2\\_24](https://doi.org/10.1007/978-3-319-59608-2_24)

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2017

# ase-PoW: a Proof of Ownership mechanism for cloud deduplication in hierarchical environments

Lorena González-Manzano<sup>1</sup>, Jose Maria de Fuentes<sup>1</sup>, and Kim-Kwang Raymond Choo<sup>2</sup>

<sup>1</sup> University Carlos III of Madrid, Leganés (Spain),  
lgmanzan, jfuentes@inf.uc3m.es,

<sup>2</sup> Department of Information Systems and Cyber Security, University of Texas at San Antonio, USA and School of Information Technology & Mathematical Sciences, University of South Australia, Australia,  
raymond.choo@fulbrightmail.org

**Abstract.** Proof-of-Ownership (PoW) can be an effective deduplication technique to reduce storage requirements, by providing cloud storage servers the capability to guarantee that clients only upload and download files that they are in possession of. In this paper, we propose an attribute symmetric encryption PoW scheme (ase-PoW) for hierarchical environments such as corporations, in which (1) the external cloud service provider is honest-but-curious and (2) there is a flexible access control in place to ensure only users with the right privilege can access sensitive files. This is, to the best of our knowledge, the first such scheme and it is built upon the ce-PoW scheme of González-Manzano and Orfila (2015). ase-PoW outperforms ce-PoW in that it does not suffer from content-guessing attacks, it reduces client storage needs and computational workload.

**Key words:** Deduplication technique, Proof of Ownership, Symmetric encryption, Access control

## 1 Introduction

Cloud storage services are increasingly popular with both individual and organizational users<sup>1</sup>. This is, perhaps, unsurprising due to the wide range of cloud storage solutions offering significant or unlimited amount of storage to individual users and organizations such as educational institutions [1, 2]. Cloud storage has also attracted the attention of researchers [3] such as forensics [4, 5, 6], security and privacy [7, 8, 9], in addition to designing efficient and effective storage solutions. For example, deduplication techniques have been the subject of recent research focus due to their potential in significant reduction of cloud storage requirements. Specifically, the deployment of deduplication techniques allows cloud

---

<sup>1</sup> <http://www.computerweekly.com/opinion/Time-to-outsource-data-storage> and <http://www.lima.co.uk/blog/3-reasons-why-businesses-choose-to-outsource-their-data-storage/>; last accessed 10 May 2016

servers to store only a single copy of the uploaded data together with the list of owners, thus, significantly reducing the storage requirements [10].

There are two main security challenges faced in deploying deduplication techniques in a hierarchical context, namely file access control and data leakage prevention. In the former, it is critical to ensure that only authorized users are granted access to the file. Let us consider a naive deduplication scheme, where a client sends a file identifier to the cloud and if it is already stored, then the server assumes that the client owns the file. However, this allows an attacker (including another malicious client) who only knows a file identifier but does not have the file to gain access to the file (e.g. by “colluding” with the file owner such as compromising the device of the file owner using malware). Proof of Ownership (PoW) scheme has been shown to be a viable solution against such an attack. PoW schemes, first introduced by Halevi *et al.* [11], guarantee that clients are in possession of the uploaded files, by presenting a proof of file ownership that can only be established when the file is available to the clients. Under a security parameter, a PoW is assumed to be secure even when an adversary knows part of the file [12]. Several PoW schemes extending the work of Halevi *et al.* [11] have been proposed in the literature. For example, Di Pietro *et al.* [12] propose the s-PoW scheme designed to enhance client-side efficiency, Blasco *et al.* [13] present the bf-PoW scheme designed to achieve flexibility and scalability, and González-Manzano *et al.* [14] propose the ce-PoW scheme designed to deal with honest-but-curious servers and to achieve efficiency. In a hierarchical deployment, deduplication also needs to ensure that users have rights to access the data and a number of proposals have been presented in this regard [15, 16, 17, 18, 19, 20, 21]. Such requirement is also referred to as authorized deduplication [19]. However, existing proposals generally impose a significant burden on the cloud server or do not necessarily ensure that users own the file.

In the data leakage prevention scenario, given that data storage is outsourced, cloud servers are assumed to be honest-but-curious. Such servers honestly execute the proposed scheme but they may attempt to learn the stored content. To mitigate such threats, previous attempts have focused on encrypting the files. We observe that current solutions generally use symmetric encryption, due to the need to ensure that the result of the encryption of a same file remains the same in order to allow deduplication. The most common approach is to apply the Convergent Encryption (CE) scheme [16], where files are encrypted using their content as a key [22]. However, CE suffers from a number of limitations including content guessing attacks (i.e. malicious clients are able to discover the plaintext content) [16, 14]. There is no known PoW solution that provides both file access control and data leakage prevention. This is the gap that this paper seeks to contribute to.

We present the Attribute Symmetric Encryption Proof of Ownership scheme (hereafter referred to as ase-PoW), which extends the ce-PoW scheme presented in [14]. Specifically, we include a lightweight access control procedure that does not impose any burden on the cloud server and our proposed scheme is designed to withstand content guessing attacks. To ensure that the scheme can be deployed

in a hierarchical application, access control is achieved through encryption where the keys are linked to user attributes. Thus, only users with a given attribute (say belonging to a particular department, e.g. human resources) can access (and further deduplicate) the corresponding file (e.g. employees’ contracts). We then demonstrate that ase-PoW outperforms ce-PoW, in terms of both storage and client efficiency.

The structure of the paper is as follows. Section 2 reviews the ce-PoW scheme. Section 3 describes the proposed scheme ase-PoW. We demonstrate the security and utility of the proposed scheme in Sections 4 and 5. Related work is discussed in Section 6. Finally, Section 7 concludes the paper and outlines future research directions.

## 2 Revisiting the ce-PoW scheme

The ce-PoW scheme [14] is an efficient PoW scheme designed for an environment involving honest-but-curious servers. In the scheme, files are encrypted using Convergent Encryption (CE) where the encryption key is the file itself, and proof of ownership is achieved by requesting from clients some CE-encrypted chunks. Specifically, let  $\mathcal{H}_1 : \{0, 1\}^* \rightarrow \{0, 1\}^n$  and  $\mathcal{H}_2 : \{0, 1\}^B \rightarrow \{0, 1\}^l$  cryptographic hash functions, where  $B$  and  $l$  represent the chunk size and the token size respectively; and  $n$  is a positive integer. There are two phases in the ce-PoW scheme, namely:

- **Initialization:** The client sends the file size to the server, which responds with the number of chunks the file should be split into. Then, the client convergently encrypts each chunk, computes  $\mathcal{H}_2$  over each encrypted chunk and finally, computes  $\mathcal{H}_1$  over the resulting hashes obtaining  $h_c$ . Both  $h_c$  and the encrypted chunks are then sent to the server. The server will compute  $h_c$  from the received encrypted chunks and verify whether the computed result is the same as the received data to avoid poisoning attacks. If the verification returns true, then the server creates an array storing three structures, namely: one structure to store the list of owners, one structure to maintain a list of challenges, and another one to store the responses to the challenges.
- **Challenge:** The server receives a  $h_c$  value. If  $h_c$  entry is not found, then the server requests the client to upload the file size; thus, reverting to the initialization phase. If an entry for  $h_c$  exists, then the server loads in memory the first unused challenge together with the corresponding responses, prior to sending the challenge to the “claiming” client. The client then computes the response token for each of the  $J$  chunk indices and sends the array of response tokens to the server. Subsequently, the server checks whether the received tokens match the stored tokens. If the check returns true, then the server labels the PoW as successful and assigns the file to the client. Otherwise, the client is considered to have failed the PoW.

This scheme is proven to be secure under the bounded leakage setting, in which a limited portion of a file may be leaked (i.e. 64MB) but the file owner is able to prove the possession of such a file in a secure manner [11].

However, the ce-PoW scheme suffers from two main weaknesses.

1. Due to the use of convergent encryption, the scheme is vulnerable to the inherent content-guessing attacks.
2. Due to the need to store decryption keys (i.e. chunk hashes) on the client devices in order to decrypt downloaded files, the number of keys stored by any client corresponds to the file size for files smaller than 64MB and 5% of the file size for files larger than 64MB. This is an unrealistic requirement, particularly for client devices such as smart phones.

### 3 Attribute Symmetric Encryption Proof of Ownership Scheme

In this section, we present an overview of the system, the threat model and the goals of the proposed scheme, prior to describing the scheme.

#### 3.1 System overview

To explain how ase-PoW can be implemented in practice, let us consider the following use case.

*Use case.* A University consists of Departments ( $D_i$ ) divided into Groups ( $G_i$ ), which work in different Projects ( $P_i$ ). Members of a given  $G_i$  may work in different  $P_i$ . In addition, each  $P_i$  has a  $G_i$  who is the designated leader. Users involved in  $D_i$ ,  $G_i$  and  $P_i$  have their own attributes and thus, they have corresponding keys.

For simplicity, we now assume that there are two departments,  $D_1$  and  $D_2$  (Figure 1). The former is composed by  $G_1$  and  $G_2$ , which manage a pair of projects,  $P_1$  and  $P_2$ .  $G_1$  and  $G_2$  are leaders of  $P_1$  and  $P_2$  respectively. In  $D_2$ , there is only one group,  $G_3$ , which is the leader of  $P_3$ . Moreover,  $G_1$  takes part in  $P_2$  and  $G_2$  is involved in  $P_3$ .

In terms of managing files  $f_1$  and  $f_2$  of  $P_3$ , there are two main steps, namely: encryption and deduplication. For encryption, let us assume that  $f_1$  needs to be accessible only to users involved in  $P_3$  whereas  $f_2$  can only be accessed by  $G_2$  members working on  $P_3$ . Thus,  $f_1$  is symmetrically encrypted with  $K_{P_3}$  while  $f_2$  is also symmetrically encrypted with  $K_{f_2}$  which is created encrypting  $K_{G_2}$  with  $K_{P_3}$ .

For deduplication, we will now focus our discussion on  $f_1$ . At first, one of the users involved in  $P_3$  uploads  $f_1$  after encryption, together with its digest  $h_c$  (recall Section 2). It must be noted that  $h_c$  is used by the server to identify  $f_1$  in subsequent uploads. Then, the server prepares three data structures, namely one to store the list of owners, another to keep a list of challenges, and the third for their expected responses. Thus, at the time other client tries to upload  $f_1$ ,

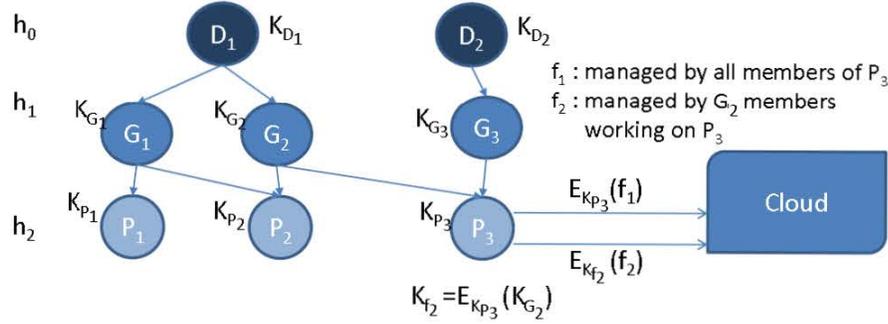


Fig. 1. An example use case

the server requests a set of challenges. If such challenges are provided as per specification, it is assumed that  $f_1$  is owned by this client and the client will be added to the list of owners.

### 3.2 Entities

In ase-PoW, three entities are identified. First, the **client** is the entity that holds the file to be deduplicated. Each client is a user of a hierarchical organizations, i.e. a corporation, that belongs to one or more of its areas, i.e. departments. He performs the PoW to the storage server, which is in charge of keeping all files. For each file, the **server** manages data structures that contain the identities of clients which are allowed to access to the file and the challenges to be satisfied in the PoW, as it will be explained later. Apart from the client and the server an **Attribute Certificate Service (ACS)** is introduced. It is responsible for managing which users belong to each department over time. As such, ACS grants or revokes the permission to access to confidential files, once a user joins or leaves an area.

### 3.3 Threat model

The adversary is assumed to be an attacker who attempts to download a file he does not possess, via the following means:

- **Content-guessing attack** where attackers intercept interchanged PoW challenges and try to guess their content.
- **Collusion attack** in which the legitimate file owner colludes with a malicious client (an adversary  $\tilde{C}$ ) leaking part of the file content. In [14], a PoW scheme works on the assumption that the exchange of information is not interactively performed along the PoW challenge, in addition to the assumption that 64MB is sufficiently large to discourage collusion [11].

### 3.4 Goals

A total of six goals are considered in this proposal. The first three goals focus on security, namely: goals one and two capture the scenario where an adversary seeking to download a file he does not own, and the third goal deals with access control. The remaining set of goals capture the performance requirements, namely: minimizing bandwidth, memory consumption and storage space. Specifically, the goals of this proposal are as follows:

**Security:** an adversary  $\tilde{C}$ , who does not possess a complete file  $f$ , has a negligible advantage in succeeding in a PoW given a security parameter  $\kappa$ .

**Collusion resistance:** an adversary  $\tilde{C}$ , who does not possess a complete file  $f$ , must exchange a minimum amount  $S_{min}$  of information with the legitimate owner of  $f$  to be successful in the PoW. According to Halevi et al. [11]  $S_{min}$  is set to 64MB.

**Simple fine-grained access control:** the encryption, apart from providing confidentiality, should allow the management of access control without involving the cloud server in the access control management process and without requiring the involvement of additional tasks for the client and the server. Besides, it should be as fine-grained as possible, thus allowing the specification of different encryption policies.

**Bandwidth efficiency:** the number of exchanged bytes between client and server along a PoW execution should be minimized.

**Server space efficiency:** in a PoW, the server should load in memory a small piece of information independent of the input file size.

**Client space efficiency:** regardless of the use of cryptography, clients have to store as few keys as possible. In addition, the number of stored keys should be independent of file sizes.

### 3.5 The Proposed Scheme

The scheme builds on the scheme presented in [14], and the key differences are the use of symmetric encryption on the chunks and the enforcement of access control.

To carry out cryptographic computations, ase-PoW leverages on the hierarchical structure of organizations as well as the existence of ACS. In particular, belonging to each organizational unit (say Department or Group) or working on a given Project implies that each user holds an attribute. Each attribute is linked to a key provided by ACS. Thus, when a user requests the attestation of attributes, ACS verifies such attributes and provides keys accordingly.

In this scheme, there are two phases – see Figure 2. In the **Initialization phase**, the client firstly requests keys to ACS and symmetric keys are delivered when the client possesses the right attributes. Then, the client requests the upload of a file sending the digest of the encrypted file  $h_c$  to the server. Once the server verifies the file is not already stored ( $h_c$  not stored), the client sends the file size and the server provides the amount of chunks the file should be split into.

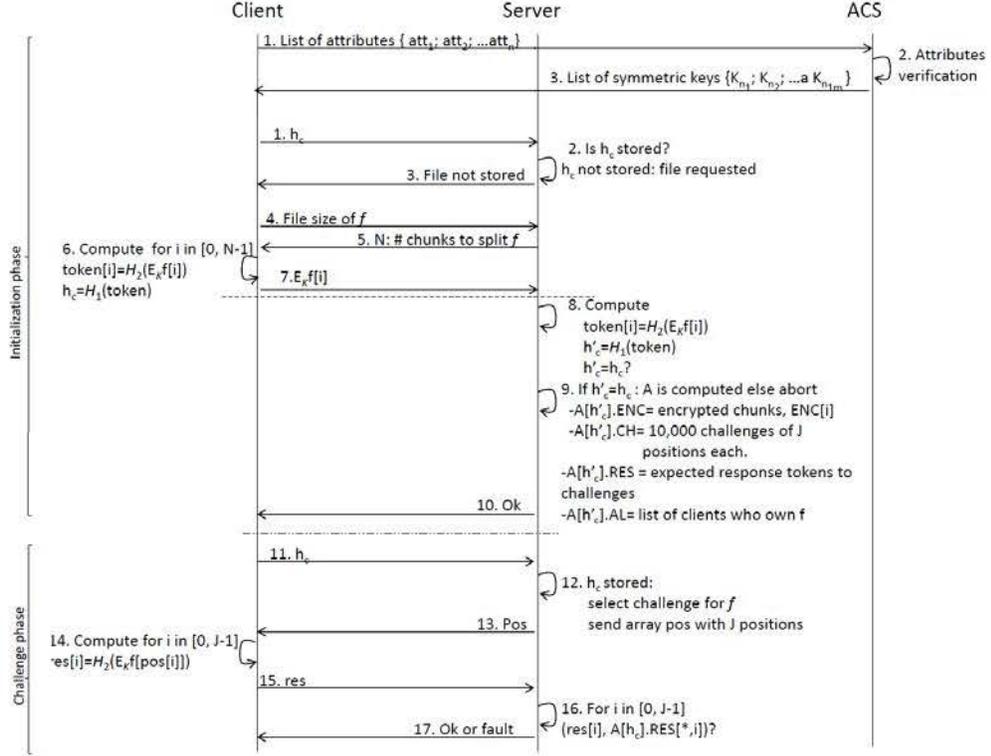


Fig. 2. ase-PoW description.

The client symmetrically encrypts each file chunk applying keys provided by ACS (Algorithm 1). In case multiple keys are at stake, each chunk is symmetrically encrypted with a key ( $K_S$ ). Such key is formed by a recursive encryption of the set of keys that are found in the path from  $f_i$  to the highest group of corporate members that need access to it. Thus, let  $a$  and  $b$  the levels in which  $f_i$  and the said group are located, respectively. The encryption key is then formed by  $K_S = E_{K_a}(\dots(E_{K_{b-1}}(K_b)))$ . Note that if  $a = b$ , then no recursive encryption is needed since the file is already accessible to the smallest group of members. After encryption, each encrypted chunk of file  $f$  is denoted as  $E_{K_S}f[i]$ . In last place, the encrypted file is sent to the server which initializes an array  $\mathcal{A}$  where  $h_c$  is the lookup key -  $\mathcal{A}[h_c].ENC$  contains encrypted file chunks,  $\mathcal{A}[h_c].CH$  stores 10,000 challenges (with  $J$  random positions each),  $\mathcal{A}[h_c].RES$  keeps the expected response tokens that correspond to the challenges and  $\mathcal{A}[h_c].AL$  contains a list of identifiers of clients who own  $f$  (Algorithm 2).

In the **Challenge phase**, when a stored file is requested because  $h_c$  sent by the client matches with the one stored in the server, the client encrypts requested file chunks and performs a digest  $\mathcal{H}_2$  over it until complete the requested challenge (Algorithm 3). As aforementioned, the encryption may involve creating  $K_S$

---

**Algorithm 1:** First client upload at client side

---

**Input:** Number of chunks  $N$ , a file  $f$  and encryption key(s)  $K_j$  in the encryption order  $\{1, \dots, j\}$   
**Output:** Hash  $h_c$  of the symmetrically encrypted file chunks; and symmetric encrypted chunks  $E_{K_j}f[i]$  or  $E_{K_j}(\dots(E_{K_1}f[i]))$   
**for**  $i = 0$  **to**  $N - 1$  **do**  
    |  $token[i] = [\mathcal{H}_2(E_{K_N}f[i]) \text{ or } \mathcal{H}_2(E_{K_j}(\dots(E_{K_1}f[i])))];$   
**end**  
 $h_c = \mathcal{H}_1(token);$   
**return**  $h_c$  and  $E_{K_j}f[i]$  or  $E_{K_j}(\dots(E_{K_1}f[i]));$

---

---

**Algorithm 2:** First client upload in ase-PoW. Server side (analogous to ce-PoW [14] except for the encryption procedure)

---

**Input:** Encrypted chunks  $ENC[i] = E_{K_S}f[i]$  and  $h_c$  uploaded by client  $\mathcal{C}$ .  
**Output:** The entry  $\mathcal{A}[h_c]$   
**for**  $i = 0$  **to**  $N - 1$  **do**  
    | Compute array  $token$  from received  $ENC[i]$   
    |  $token[i] \leftarrow \mathcal{H}_2(ENC[i]);$   
**end**  
 $h_c \leftarrow \mathcal{H}_1(token);$   
**if**  $\neg Match(h_c, \mathcal{H}_1(token))$  **then**  
    | **return**  $\perp$ ;  
**end**  
Store 10,000 random challenges  $CH$  with  $J$  indexes each  
**for**  $x = 0$  **to** 9999 **do**  
    | **for**  $y = 0$  **to**  $J - 1$  **do**  
        |  $pos[y] = PRF(seed); CH[x, y] = pos[y]; RES[x, y] = token[pos[y]];$   
    | **end**  
**end**  
 $\mathcal{A}[h_c].ENC \leftarrow ENC; \mathcal{A}[h_c].CH = CH; \mathcal{A}[h_c].RES = RES; \mathcal{A}[h_c].AL = \{id(\mathcal{C})\};$   
**return**  $\mathcal{A}[h_c];$

---

recursively. Finally, the PoW will be passed or not according to the verification the server performs comparing the received responses ( $res$ ) with the stored ones ( $\mathcal{A}[h_c].RES$ ) (Algorithm 4).

## 4 Security Analysis

We now demonstrate that the ase-PoW scheme achieves the first three goals described in Section 3.4, and the remaining goals will be addressed in Section 5.

---

**Algorithm 3:** Challenge phase at client side

---

**Input:** A file  $f$ , an array  $pos$  of  $J$  indexes and encryption key(s)  $K_j$  in the encryption order  $\{1; \dots; j\}$   
**Output:** An array  $res$  of  $J$  response tokens  
**for**  $i \leftarrow 0$  **to**  $J - 1$  **do**  
     $res[i] \leftarrow [\mathcal{H}_2(E_{K_j} f[i]) \text{ or } \mathcal{H}_2(E_{K_j}(\dots(E_{K_1} f[i])))];$   
**end**  
**return**  $res$ ;

---

---

**Algorithm 4:** Challenge phase in ase-PoW. Server side. (Analogous to ce-PoW [14])

---

**Input:**  $h_c$  of a file  $f$ ; two arrays  $pos$  and  $res$  of  $J$  indexes and client response tokens, respectively  
**Output:** The outcome of the challenge  
**for**  $i \leftarrow 0$  **to**  $J - 1$  **do**  
    **if**  $\neg Match(res[i], \mathcal{A}[h_c].RES[*, i])$  **then**  
        **return**  $\perp$ ;  
    **end**  
**end**  
 $\mathcal{A}[h_c].AL \leftarrow \mathcal{A}[h_c].AL \cup \{id(C)\};$   
**return**  $\top$ ;

---

#### 4.1 Security

The security analysis of ase-PoW is based on ce-PoW [14] and builds on the earlier proofs of Di Pietro et al. [12] and Blasco et al. [13]. The adversary is challenged on  $J$  independent chunk positions where the probability of success is:

$$P(succ) = P(tok_i)^J = (p + 0.5^l(1 - p))^J, \quad (1)$$

where  $p$  is the probability that a malicious client  $\tilde{C}$  knows part of the file; thus, able to perform a collusion or a content guessing attack. From Eq. 1, a lowerbound for  $J$  is derived which ensures  $P(succ) \leq 2^{-\kappa}$ , where  $\kappa$  is the security parameter, as

$$J \geq \frac{\kappa \ln 2}{(1 - p)(1 - (0.5^l))} \quad (2)$$

In this regard, the first goal of ase-PoW, *security*, is satisfied when Equation 2 holds under parameter  $\kappa$ . Moreover, the second requirement, *collusion resistance* involves ensuring that a legitimate client  $C$  does not exchange  $S_{min}$  bytes with a malicious client  $\tilde{C}$  to allow the malicious client to run a successful PoW for an unknown file. Considering that chunks are managed, there are  $\frac{F}{B}$  tokens in a file  $f$  of size  $F$  of chunks of size  $B$ , the token length  $l$  can be set as:

$$l \geq S_{min} \frac{B}{F} \quad (3)$$

The third security goal, *simple access control*, is also achieved. Access control is enforced by the fact that the ownership of attributes becomes a key to access files. Just an ACS is introduced to deliver keys once attributes are attested and it does not have any active role in the deduplication process. In addition, fine grained access control is achieved due to the use of recursive encryptions. They can be compared with the encryption of files with attributes (keys) concatenated with AND operators, meaning that different encryption policies can be applied.

Apart from the previous issues, ase-PoW tackles the content-guessing attack. In contrast to [12] and [13] which do not apply encryption and to [14] which applies CE, a symmetric encryption scheme is applied herein. It ensures encrypted chunks are independent of files entropy, thus preventing this attack. Indeed, even if files were obtained by attackers they could not be decrypted.

## 4.2 Complexity

We now evaluate the complexity of ase-PoW with that of Di Pietro et al. [12], Blasco et al. [13] and González-Manzano et al. [14]. In particular, the evaluations (see Table 1) focus on client and server computation and I/O, server memory usage, bandwidth, and number of used keys (if required).

We remark that ase-PoW complexity differs from ce-PoW in a critical aspect, namely: the number of keys managed by the client. Particularly, in ase-PoW, client computation involves a symmetric encryption scheme based on a chosen number of recursive encryptions ( $n_{re}$ ) in relation to owned attributes. As a result, the client only needs to manage up to  $n_{re}$  keys, regardless of the number of files under deduplication. In ce-PoW, the client needs to store all chunk hashes of every file deduplicated, as these hashes are the file-specific decryption key. This may be up to 5% the file size, e.g. 50MB for 1GB files. Thus, ase-PoW reduces the storage space needed in the client to allow deduplication.

The comparisons between ase-PoW and s-PoW and bf-PoW are similar to those with ce-PoW. It is clear that client and server computations involve less complexity in s-PoW and bf-PoW than in the other schemes, since there is no encryption involved. The bandwidth requirement is also noticeably lower in s-PoW and bf-PoW, as in ase-PoW and ce-PoW  $J$  tokens are sent to the server. However, neither s-PoW nor bf-PoW protect against honest-but-curious servers. Thus, striking a balance between security and efficiency is expected. The remaining set of features can be considered similar among all studied schemes.

## 5 Performance Evaluation

We now present the findings of our evaluations, based on the settings described in [14]. Specifically, ase-PoW is evaluated against bf-PoW [13] and ce-PoW [14]. We did not evaluate our proposal against s-PoW [12] because it has been shown that ce-PoW outperforms s-PoW.

All schemes were implemented in C++ using OpenSSL as a cryptographic library. AES in counter mode and SHA-1 are the two main operations. As in

**Table 1.** Complexity comparative summary. Applied symbols are taken from Section 4

	s-PoW [12]	bf-PoW [13]	ce-PoW	ase-PoW [14]
Client computation	$O(F) \cdot hash$	$O(F) \cdot hash$	$O(B) \cdot CE \cdot hash \cdot hash$	$O(B) \cdot Sym. \cdot n_e \cdot hash$
Client I/O	$O(F)$	$O(F)$	$O(F)$	$O(F)$
Server init computation	$O(F) \cdot hash$	$O(F) \cdot hash$	$O(B) \cdot hash \cdot hash$	$O(B) \cdot hash$
Server regular computation	$O(n \cdot \kappa) \cdot PRF$	$O\left(\frac{l \cdot \kappa \cdot (\log(1/p_f) \cdot l)}{p_f}\right) \cdot hash$	$O(n \cdot l \cdot \kappa) \cdot PRNG$	$O(n \cdot l \cdot \kappa) \cdot PRNG$
Server init I/O	$O(F)$	$O(F)$	$O(F)$	$O(F)$
Server regular I/O	$O(n \cdot \kappa)$	$O(0)$	$O(0)$	$O(0)$
Server memory usage	$O(n \cdot \kappa)$	$O\left(\frac{\log(1/p_f)}{l}\right)$	$O(n \cdot l \cdot \kappa)$	$O(n \cdot l \cdot \kappa)$
Bandwidth	$O(\kappa)$	$O\left(\frac{l \cdot \kappa}{p_f}\right)$	$O(l \cdot \kappa)$	$O(l \cdot \kappa)$
# stored keys	-	-	up to 5% file size	att

[14],  $\mathcal{H}_1$  and  $\mathcal{H}_2$  are based on SHA-1 being  $\mathcal{H}_2$  applied over encrypted chunks extending the length of the hash to  $l$  through the use of the stream cipher RC4.

To ensure a fair evaluation, we used the parameters defined in [13, 14], namely: the security parameter is set to  $\kappa=66$ ;  $S_{min}=64\text{MB}$ , the size of tokens ( $l$ ) is set to  $\{16, 64, 256, 1024\}$  bytes, the probability ( $p$ ) that an adversary knows a chunk of a file is set to  $\{0.3; 0.5; 0.75; 0.95\}$  and that key size is 256B. According to these values and Equation 2, the number of requested challenges ( $J$ ) corresponds to  $\{65, 91, 182, 914\}$ . Similarly, considering the said values of  $l$ ,  $S_{min}$  and the input file size, the size of chunks ( $B$ ) is according to Equation 3 – see Table 2.

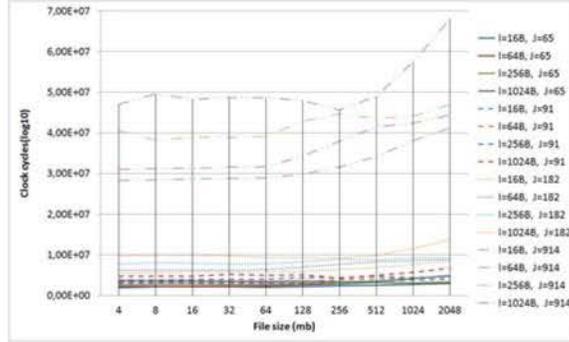
**Table 2.** Chunk sizes ( $B$ ) in bytes, computed from the file size, the token size and  $S_{min}$

l(B)	File size (MB)									
	4	8	16	32	64	128	256	512	1024	2048
<b>16</b>	16	16	16	16	16	32	64	128	256	512
<b>64</b>	64	64	64	64	64	128	256	512	1024	2048
<b>256</b>	256	256	256	256	256	512	1024	2048	4096	8192
<b>1024</b>	1024	1024	1024	1024	1024	2048	4096	8192	16384	32768

The experiments were performed on a AMD Athlon(tm) II x2 220 processor with 4GB of RAM. Input files were randomly generated and their sizes ranged from 4MB to 2GB doubling the size at each step.

The client side computation is studied in the following section. Server side computation of ase-PoW is similar to that of ce-PoW since the server tasks remain unaltered. Then, server side computation in ase-PoW (and ce-PoW) is comparable with that of bf-PoW (see [14]).

On the client side, the most relevant issue to consider is the time taken (in clock cycles units) to compute challenges. First of all, the time taken to create the chunk encryption key were computed, resulting in 12262 clock cycles when there were 7 keys (6 recursive encryptions) and 18743 clock cycles when there were 11 keys (10 recursive encryptions). These values are considered negligible relative to the remaining part of the scheme. Then, to present a worst case



**Fig. 3.** Client response creation clock cycles for  $J=\{65,91,182,914\}$  challenges and 11 keys.

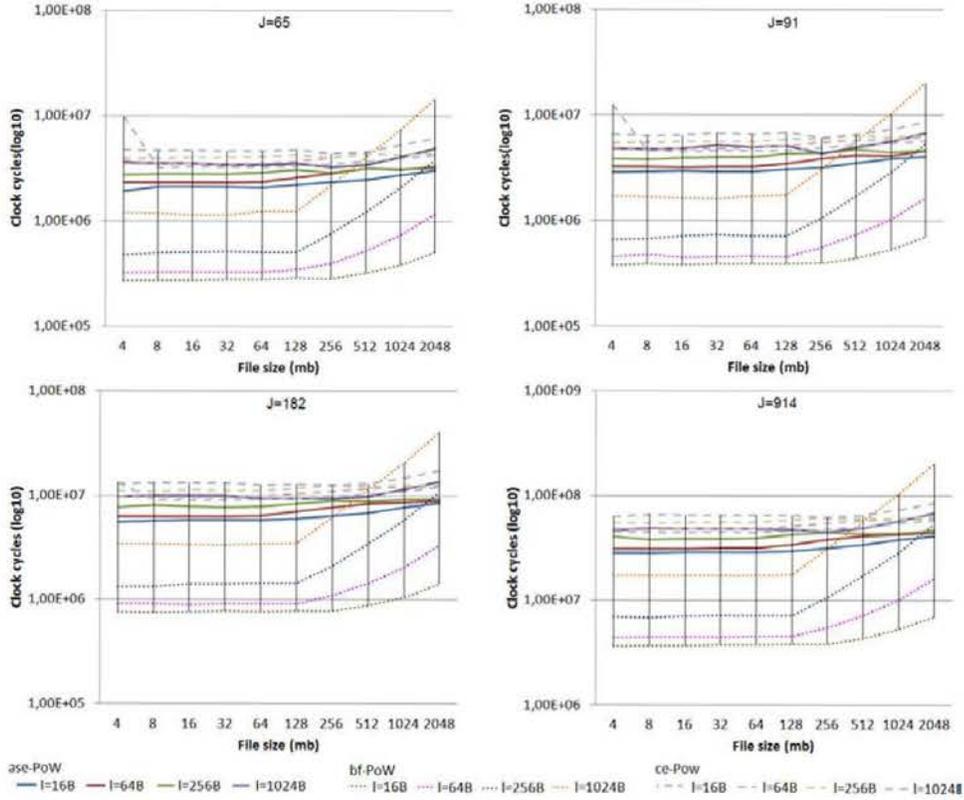
analysis, the computation of challenges considering 11 keys was studied. Results are depicted in Figure 3. We concluded that:

- The time remains constant regardless of the number of requested challenges, namely for  $J=65$  and  $J=91$ . As expected, the time increases when more challenges are computed, specially for  $J=914$ .
- The time also remains constant regardless of the file and chunk size when  $l < 64B$ . In case of higher  $l$ , e.g.  $l=1024B$ , the time increases between files of 512MB and 2 GB but it is just particularly noticeable for  $J=914$ , and to a lesser extent for  $J=182$ .

Figure 4 describes the evaluation findings of bf-PoW, ce-PoW and ase-PoW when there were 11 keys, and it is clear that bf-PoW has the best performance. This is because in bf-PoW, a token is computed for each  $J$  chunk index through a hash function. However, performance of bf-PoW is comparable with ase-PoW for  $l=1024b$  and bf-PoW does not protect against honest-but-curious servers. On the other hand, ase-PoW outperforms ce-Pow in all cases. In ce-PoW, a hash per encrypted chunk is computed which increases the computation time. By contrast, ase-PoW symmetrically encrypts chunks and though the chunk encryption key may involve several recursive encryptions, findings demonstrate that the time to compute this key is negligible in comparison with the rest of the process.

## 6 Related Work

Deduplication, such as PoW schemes, has been the subject of research focus [23, 11, 12, 13, 14]. For example, the PoW schemes in [24, 25, 26, 15, 17, 27] are designed to work with honest-but-curious servers. Due to the use of CE in many existing PoW schemes such as [25, 26, 16, 17, 22], these schemes are not secure against content analysis attacks as previously discussed [16]. A number of proposals to avoid such pitfalls has also been proposed in recent years. For



**Fig. 4.** Client response creation clock cycles for  $J=\{65,91,182,914\}$  challenges and 11 keys.

example, in [27], files are asymmetrically encrypted and decryption keys are interchanged among clients. The proposed approach in [24] involves an identity provider designed to prevent sybil attacks, and an indexing server to prevent data leakage.

The significant amount of data managed by cloud servers necessitates the implementation of access control solutions, as this will allow servers the capability to determine whether a requesting client has the appropriate access rights. Attribute based encryption (ABE) is one commonly used method to achieve fine grained access control in the cloud [28, 29, 30, 31], where files are encrypted under a set of attributes and decrypted by a key with the right attribute policy [28]. However, deduplication is not considered in these works which is unsurprisingly since ABE is a non-deterministic encryption scheme and therefore, cannot be applied to this context.

Table 3 compares the proposed ase-PoW scheme and other similar deduplication and access control schemes, based on the use of a PoW scheme, security against honest-but-curious servers, involvement of third parties, theoretical se-

curity analysis, search of bandwidth, server and client side space efficiency, and empirical performance analysis.

**Table 3.** Comparative summary

Proposals	PoW scheme	Honest-but-curious servers	Third parties	Theoretical security analysis	Bandwidth efficiency	Server space efficiency	Client space efficiency	Empirical analysis
[15]	✓	Not specified. Public keys used	Intermediator	-	-	-	-	-
[16]	-		Metadata manager, additional server	-	-	-	-	✓
[17] [21]	✓*	Message-lock encrypt	Key server	-	✓	-	-	✓
[19]	✓*	Convergent encryption	Private cloud	-	-	-	-	-
[20]	-	Convergent encryption	Multiple servers	-	-	-	-	✓
[18]	-	Convergent encryption	Distributed key server	-	-	-	-	✓
[32]	✓	-	Auditor	✓	-	-	-	-
<b>ase-PoW</b>	✓	Symmetric encryption	Attribute certification server	✓	✓	✓	✓	✓

\*: mentioned but not applied

It is clear that access control and deduplication require additional entities and additional management tasks. In [16], for example, a metadata manager enforces key management and handles deduplication. In [17, 21], there exists a key server per group of clients which is in charge of key management and helps in the deduplication process. In [20], the Dekey scheme shares encryption keys among clients via distributed key servers. The SecDep scheme in [18] involves multiple key servers, which are also tasked with deduplication. In [32], an auditor verifies the integrity of data in the cloud.

There are only a small number of proposals using a PoW scheme while managing deduplication and access control [15, 17, 21, 19]. Although [15, 17, 21] mention the use of PoW, no concrete details are provided. In [19], an intermediary becomes the PoW verifier.

[32] appears to be the only work that provides a security analysis and no other studies examine server and client space efficiency. With the exception of [15], key storage is externalized to additional servers, relieving clients from the burden of managing and storing keys. Bandwidth efficiency is considered only in [17, 21], and just some schemes do an empirical analysis [16, 17, 21, 20, 18].

In summary, it is clear that achieving both effective and secure PoW and access control management for deduplication in the presence of honest-but-curious servers is an understudied topic.

## 7 Conclusion

Cloud storage is a trend that is unlikely to go away anytime soon, and one of the key challenges is to reduce storage requirements. Deduplication schemes are a viable solution, and in this paper, we present the Attribute Symmetric Encryption Proof of Ownership (ase-PoW) scheme. The scheme is based on recursively and symmetrically encrypting file chunks to prove the possession of

files. We demonstrate the security of the scheme, as well as the utility of the scheme using empirical analysis. Specifically, we show that ase-PoW is efficient and has better performance compared with similar schemes in the literature (e.g. outperforms ce-PoW, and ase-PoW has the benefit of having a constant computation time for file types when  $l < 64B$ ).

Future work includes enhancing access control expressiveness, and combining deduplication and Proof of Works [33] and Remote Data Possession Checking [34] in an environment with untrusted cloud servers. Moreover, simple fine-grained access control is achieved in this work but the development of a more complex access control management scheme, e.g. [30], is the following step.

## 8 Acknowledgments

This work was partially supported by the MINECO grant TIN2013-46469-R (SPINY: Security and Privacy in the Internet of You) and the CAM grant S2013/ICE-3095 CIBERDINE-CM (CIBERDINE: Cybersecurity, Data, and Risks) funded by Madrid Autonomous Community and co-funded by European funds. L. González and J. M. de Fuentes were also supported by the Programa de Ayudas para la Movilidad of Carlos III University of Madrid, Spain.

## References

1. R. Girardi, New Drive for Education with Unlimited Storage, <http://google.umich.edu/tech-updates/newdriveforeducationwithunlimitedstorage> (2014).
2. D. Quick, K.-K. R. Choo, Impacts of increasing volume of digital forensic data: A survey and future research challenges, *Digital Investigation* 11 (4) (2014) 273–294.
3. D. Zisis, D. Lekkas, Addressing cloud computing security issues, *Future Generation Computer Systems* 28 (3) (2012) 583–592.
4. B. Martini, K.-K. R. Choo, Cloud storage forensics: owncloud as a case study, *Digital Investigation* 10 (4) (2013) 287–299.
5. D. Quick, B. Martini, K.-K. R. Choo, *Cloud Storage Forensics*, Syngress Publishing / Elsevier, 2013.
6. D. Quick, K.-K. R. Choo, Dropbox analysis: Data remnants on user machines, *Digital Investigation* 10 (1) (2013) 3–18.
7. L. Li, R. Lu, K.-K. R. Choo, A. Datta, J. Shao, Privacy-preserving outsourced association rule mining on vertically partitioned databases, *IEEE Transactions on Information Forensics and Security*.
8. X. Liu, K.-K. R. Choo, R. H. Deng, R. Lu, J. Weng, Efficient and privacy-preserving outsourced calculation of rational numbers, *IEEE Transactions on Dependable and Secure Computing*.
9. X. Liu, R. H. Deng, K.-K. R. Choo, J. Weng, An efficient privacy-preserving outsourced calculation toolkits with multiple keys, *IEEE Transactions on Information Forensics and Security*.
10. D. Harnik, B. Pinkas, A. Shulman-Peleg, Side channels in cloud services: Deduplication in cloud storage, *Security & Privacy*, *IEEE* 8 (6) (2010) 40–47.

11. S. Halevi, D. Harnik, B. Pinkas, A. Shulman-Peleg, Proofs of ownership in remote storage systems, in: Proceedings of the 18th ACM conference on Computer and communications security, ACM, 2011, pp. 491–500.
12. R. Di Pietro, A. Sorniotti, Boosting efficiency and security in proof of ownership for deduplication, in: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ACM, 2012, pp. 81–82.
13. J. Blasco, A. Orfila, R. D. Pietro, A. Sorniotti, A tunable proof of ownership scheme for deduplication using bloom filters, in: Proceedings of the IEEE Conference on communications and network security, CNS, 2014.
14. L. González-Manzano, A. Orfila, An efficient confidentiality-preserving proof of ownership for deduplication, *Journal of Network and Computer Applications* 50 (2015) 49–59.
15. X. Jin, L. Wei, M. Yu, N. Yu, J. Sun, Anonymous deduplication of encrypted data with proof of ownership in cloud storage, in: Proceedings of the 13th IEEE/CIC International Conference on Communications in China (ICCC), 2013, pp. 224–229.
16. P. Puzio, R. Molva, M. Önen, S. Loureiro, Block-level de-duplication with encrypted data, *Open Journal of Cloud Computing (OJCC)* 1 (1) (2014) 10–18.
17. M. Bellare, S. Keelveedhi, T. Ristenpart, Dupless: server-aided encryption for deduplicated storage, in: Proceedings of the 22nd USENIX conference on Security, USENIX Association, 2013, pp. 179–194.
18. Y. Zhou, D. Feng, W. Xia, M. Fu, F. Huang, Y. Zhang, C. Li, Secdep: A user-aware efficient fine-grained secure deduplication scheme with multi-level key management, in: Mass Storage Systems and Technologies (MSST), 2015 31st Symposium on, 2015, pp. 1–14.
19. J. Li, Y. K. Li, X. Chen, P. P. Lee, W. Lou, A hybrid cloud approach for secure authorized deduplication, *Parallel and Distributed Systems, IEEE Transactions on* 26 (5) (2015) 1206–1216.
20. V. Naresh, G. Kumar, A novel secure deduplication implementation with efficient and reliable, *Innovative Technologies, International Journal of* 3 (6) (2015) 1001–1006.
21. M. Miao, J. Wang, H. Li, X. Chen, Secure multi-server-aided data deduplication in cloud computing, Elsevier, 2015.
22. M. W. Storer, K. Greenan, D. D. Long, E. L. Miller, Secure data deduplication, in: Proceedings of the 4th ACM international workshop on Storage security and survivability, ACM, 2008, pp. 1–10.
23. F. M. Arbour, M. M. Reker, System and method for exporting data directly from deduplication storage to non-deduplication storage, uS Patent App. 11/731,178 (Mar. 30 2007).
24. J. Stanek, A. Sorniotti, E. Androulaki, L. Kencl, A secure data deduplication scheme for cloud storage, Tech. rep., IBM (2013).
25. J. Xu, J. Zhou, Leakage resilient proofs of ownership in cloud storage, revisited, in: I. Boureanu, P. Owesarski, S. Vaudenay (Eds.), *Applied Cryptography and Network Security*, Vol. 8479 of Lecture Notes in Computer Science, 2014, pp. 97–115.
26. J. Li, X. Chen, M. Li, J. Li, P. Lee, W. Lou, Secure deduplication with efficient and reliable convergent key management, *Parallel and Distributed Systems, IEEE Transactions on* 25 (6) (2014) 1615–1625.
27. W. K. Ng, Y. Wen, H. Zhu, Private data deduplication protocols in cloud storage, in: Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12, ACM, 2012, pp. 441–446.

28. S. Yu, C. Wang, K. Ren, W. Lou, Achieving secure, scalable, and fine-grained data access control in cloud computing, in: INFOCOM, 2010 Proceedings IEEE, Ieee, 2010, pp. 1–9.
29. G. Wang, Q. Liu, J. Wu, Hierarchical attribute-based encryption for fine-grained access control in cloud storage services, in: Proceedings of the 17th ACM conference on Computer and communications security, ACM, 2010, pp. 735–737.
30. S. S. Chow, A framework of multi-authority attribute-based encryption with outsourcing and revocation, in: Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies, ACM, 2016, pp. 215–226.
31. Y. Yang, J. K. Liu, K. Liang, K.-K. R. Choo, J. Zhou, Robust access control framework for mobile cloud computing network, in: European Symposium on Research in Computer Security, Vol. 9327 of Lecture Notes in Computer Science, 2015, pp. 146–166.
32. Y. Shin, D. Koo, J. Hur, J. Yun, Secure proof of storage with deduplication for cloud storage systems, Multimedia Tools and Applications (2015) 1–16.
33. B. Laurie, R. Clayton, Proof-of-work” proves not to work; version 0.2, in: Workshop on Economics and Information, Security, 2004.
34. L. Chen, Using algebraic signatures to check data possession in cloud storage, Future Generation Computer Systems 29 (7) (2013) 1709–1715.