© Springer Berlin / Heidelberg, Lecture Notes in Computer Science (2012)
http://link.springer.com/chapter/10.1007%2F978-3-642-30436-1_17

Support for write privileges on outsourced data

Sabrina De Capitani di Vimercati[1], Sara Foresti[1], Sushil Jajodia[2], Stefano Paraboschi[3], and Pierangela Samarati[1]

[1] DTI - Università degli Studi di Milano, 26013 Crema, Italia
firstname.lastname@unimi.it
[2] CSIS - George Mason University, Fairfax, VA 22030-4444, USA
jajodia@gmu.edu
[3] DIIMM - Università degli Studi di Bergamo, 24044 Dalmine, Italia
parabosc@unibg.it

Abstract. In the last years, data outsourcing has received an increasing attention by the research community thanks to the benefits that it brings in terms of data management. A basic requirement in such a scenario is that outsourced data be made accessible only to authorized users, that is, no unauthorized party (including the storing server) should have access to the data. While existing proposals provide a sound basis for addressing such a need with respect to data dissemination (i.e., enforcement of read authorizations), they fall short on the support of write authorizations. In this paper we address such an open problem and present an approach to enforce write privileges over outsourced data. Our work nicely extends and complements existing solutions, and exploiting key derivation tokens, hashing, and HMAC functions provides efficient and effective controls.

Keywords: Data outsourcing, data protection, authorization management

1 Introduction

Data outsourcing offers to end users and companies the opportunity to benefit from the lower costs, higher availability and larger elasticity that are offered by the rapidly growing market of cloud providers. The major obstacle to the adoption of cloud storage services is commonly recognized to be the uncertainty about a correct management of security requirements, with the user interested in robust guarantees about the confidentiality and integrity of the outsourced data. Several techniques have been recently proposed by the research community [5, 10, 16]. Most of these proposals have been developed with reference to scenarios where the data should remain confidential also to the external server storing them, which is considered *honest-but-curious*, i.e., trustworthy for managing resources but should not see or learn the resource content. To provide such confidentiality guarantee, these proposals (e.g., [5, 10, 16]) assume data to be encrypted before being outsourced to the external server, and they associate with the encrypted data additional indexing information that can be used by the server to perform queries on the encrypted data. For efficiency reasons, encryption is based on symmetric keys. Earlier proposals typically consider data to be encrypted with a single key, assuming then all users to have complete visibility of the resources in the data collection or the data owner to mediate access requests to the data to enforce write authorizations. More recent proposals, addressing the problem of providing selective visibility over the data to the users (different sets of users can enjoy visibility over different resources), have proposed the application of a 'selective encryption' approach. Intuitively, different user views can be enforced by using different encryption keys for the resources and users will be able to have visibility on the resources depending on the keys they know. Proper modeling and derivation techniques have been devised to ensure limited key management overhead in such selective encryption.

While interesting and promising, however, all the solutions above assumed outsourced data to be read only. In other words, the owner can modify resources while all other users can only read them. Such an assumption can result restrictive in several scenarios where a data owner outsourcing the data to the external server may also want to authorize other users (again selectively) to write the resources it stores. We then build upon and complement previous proposals providing a solution for enforcing write authorizations on the encrypted outsourced data. Our solution is based on the same principles as previous proposals exploiting encryption itself for enforcing access control and having efficiency and manageability in mind. Our proposal enjoys compatibility and easy deployment with previous proposals, thus providing a natural and efficient extension to them for enforcing write authorizations.

The remainder of the paper is organized as follows. Section 2 introduces the basic concept on previous work on which our proposal in based. Section 3 illustrates our proposal for enforcing write authorizations. Section 4 discusses the control features of our proposal that allow a data owner to check the write operations executed and detect possible misbehaviors by the server or by the users. Section 5 discusses related work. Finally, Section 6 concludes the paper.

2 Basic concepts

Our work builds upon and extends a previous proposal [8] for confidential data outsourcing. In this section, we briefly introduce the basic concepts of the original proposal that are useful to our treatment.

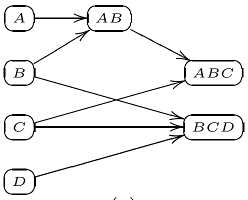
According to the proposal in [8], a data owner outsourcing data to a honest-but-curious server and wishing to provide selective visibility over them to other users encrypts resources before outsourcing them to the server and reflects the authorization policy in the encryption itself. Therefore, each resource o is encrypted with a key to be made known only to the users authorized to read o, that is, to users who belong to the access control list of o. Symmetric encryption is used and different keys are assumed: one for each user and one for each group of users that corresponds to an access control list. The adoption of a *key derivation technique* based on public tokens permits users to access the system while

o	**acl**
o_1	ABC
o_2	BCD
o_3	ABC
o_4	AB

(a)

r_label	**o_id**	**encr_resource**
l_{ABC}	1	zKZlJxVcCC0g
l_{BCD}	2	t9qdJqC7ImXU
l_{ABC}	3	AxalPH8v37Ts
l_{AB}	4	xwfPJSn.MVqY

(b)



$$\begin{tikzcd}
A \arrow[r] & AB \arrow[dr] & \\
B \arrow[ur] \arrow[drr] & & ABC \\
C \arrow[urr] \arrow[rr] & & BCD \\
D \arrow[urr] & &
\end{tikzcd}$$

(c)

$\mathbf{l_i}$	$\mathbf{l_j}$	$\mathbf{t_{i,j}}$
l_A	l_{AB}	$k_{AB} \oplus h(k_A, l_{AB})$
l_B	l_{AB}	$k_{AB} \oplus h(k_B, l_{AB})$
l_B	l_{BCD}	$k_{BCD} \oplus h(k_B, l_{BCD})$
l_C	l_{ABC}	$k_{ABC} \oplus h(k_C, l_{ABC})$
l_C	l_{BCD}	$k_{BCD} \oplus h(k_C, l_{BCD})$
l_D	l_{BCD}	$k_{BCD} \oplus h(k_D, l_{BCD})$
l_{AB}	l_{ABC}	$k_{ABC} \oplus h(k_{AB}, l_{ABC})$

(d)

Fig. 1. An example of four resources with their acls (a), encrypted resources (b), key derivation graph (c), and tokens (d)

having to manage only one key. Each key k_i is identified by a public label l_i. Given keys k_i and k_j, token $t_{i,j}$ is computed as $k_j \oplus h(k_i, l_j)$, with $\oplus$ the bitwise `xor` operator, and h a deterministic cryptographic function. Token $t_{i,j}$ permits to derive key k_j from the knowledge of key k_i and public label l_j [2]. All keys with which resources are encrypted are then connected in a *key derivation graph*. A key derivation graph is a DAG whose nodes correspond to keys (of users and acls) and whose edges correspond to tokens that ensure that each user can - via a sequence of public tokens - derive the keys corresponding to the sets to which she belongs. Each users is then communicated the key of the node representing herself in the graph. Each resource is encrypted with the key corresponding to its acl. Encrypted resources as well as the tokens are outsourced to the server. In particular, for each resource o, the external server stores the encrypted version of the resource together with the resource identifier and the label of the key with which the resource is encrypted. A user authorized to read a resource (i.e., belonging to its acl) can, via the tokens available on the server, derive the key corresponding to the acl of the resource and decrypt it.

Example 1. Consider a system with four users $\mathcal{U}=\{A,B,C,D\}$ and four resources $\mathcal{O}=\{o_1,o_2,o_3,o_4\}$, whose acls are reported in Figure 1(a). Figure 1(b) illustrates the encrypted resources stored at the server, where: *r_label* is the label of the key used to encrypt the resource (i.e., the key associated with its acl); *o_id* is the resource identifier; and *encr_resource* is the encrypted resource. Figure 1(c) illustrates the key derivation graph enforcing the authorizations. For simplicity and readability, in the key derivation graph we denote a key corresponding to a given acl U (i.e., a key with label l_U and value k_U) simply with U. Figure 1(d) illustrates the tokens corresponding to the key derivation graph in Figure 1(c).

3 Enforcement of write authorizations

The support of only read accesses may result limiting when considering emerging data sharing scenarios (e.g., document sharing), where users different from the data owner can be authorized to modify resources. Unfortunately, the keys associated with resources for regulating the read accesses to them cannot be used for regulating write accesses as well. As a matter of fact, we can imagine that in many situations the set of users authorized to write a resource is different from (typically being a subset of) the set of users authorized to read the resource. A straightforward solution for enforcing write authorizations can be to simply outsource to the external server the authorization policy (for write privileges) as is and have the server to perform traditional (authorization-based) access control. This would involve user authentication and policy enforcement, with the drawback of requesting a considerable management overhead. Following the same spirit of the proposal in [8], and to the aim of providing an extension easily integrable with it, we aim at basing enforcement of write authorizations also on encryption. While simply encrypting a resource with a key known to all and only the users authorized to read the resource automatically ensures enforcement of read authorizations, enforcement of write privileges requires cooperation from the external server. Having resources being tied to access restrictions to them by means of cryptographic solutions provides more robustness and flexibility of the control, whose enforcement is less exposed to server misbehaviors and not affected by possible server allocation strategies.

The basic idea of our approach consists in associating each resource with a *write tag*, which is defined by the data owner, and encrypting it with a key to be known only to the users authorized to write the resource and to the external server (i.e., only the external server and the users authorized to write a resource can have access to the corresponding write tag). The server will accept a write operation on a resource when the user requesting it shows knowledge of the write tag. The key used for encrypting the write tag has to be shared among the server and the writers, and we leverage on the underlying structure already in place for regulating read operations to achieve this.

3.1 Key derivation structure

Elaborating the approach in [8], and adapting it to our context, we introduce a *set-based key derivation graph* as follows.

Definition 1 (Set-based key derivation graph). *Let $\mathcal{U}$ be a set of users and $\mathcal{U}' \subseteq 2^{\mathcal{U}}$ be a family of subsets of users in $\mathcal{U}$ such that $\forall u \in \mathcal{U}$, $\{u\} \in \mathcal{U}'$. A* set-based key derivation graph *is a triple $\langle \mathcal{K}, \mathcal{L}, \mathcal{T} \rangle$, with $\mathcal{K}$ a set of keys, $\mathcal{L}$ the set of corresponding labels, and $\mathcal{T}$ a set of tokens, such that:*

1. *$\forall U \in \mathcal{U}'$, there exists a key $k_U \in \mathcal{K}$;*
2. *$\forall u \in \mathcal{U}$, $\forall U \in \mathcal{U}' - \{u\}$, there exists a token $t_{u,U}$ or a sequence $\langle t_{u,U_1}, \ldots, t_{U_n,U} \rangle$ of tokens in $\mathcal{T}$, with $t_{c,d}$ following $t_{a,b}$ in the sequence if $b = c$, and $n \geq 1$, iff $u \in U$.*

Intuitively, the approach in [8], while not explicitly reporting the definition above, basically translates to it if we assume $\mathcal{U}'$ to include, besides singleton sets of users, the read acls of resources.

Since in the scenario we consider each resource is associated with a write tag that must be encrypted with a key shared among the server and the authorized

```
INPUT
$\mathcal{U}$ : set of users
$\mathcal{S}$ : external server
$\mathcal{U}' \subseteq 2^{\mathcal{U}}$ : family of subsets of users in $\mathcal{U}$
$\mathcal{U}'' \subseteq \mathcal{U}'$ : subset of $\mathcal{U}'$
OUTPUT
$\langle \mathcal{K},\mathcal{L},\mathcal{T} \rangle$ : key derivation structure
DEFINE_KEY_DERIVATION_STRUCTURE
/* Step 1: define the set-based key derivation graph */
$\mathcal{K}' := \emptyset$, $\mathcal{L}' := \emptyset$; $\mathcal{T}' := \emptyset$
for each $U \in \mathcal{U}'$ do /* generate a key for all $U \in \mathcal{U}'$ (C1 in Def. 1) */
  generate a key $k_U$ and a label $l_U$
  $\mathcal{K}' := \mathcal{K}' \cup \{k_U\}$, $\mathcal{L}' := \mathcal{L}' \cup \{l_U\}$
/* define a set of tokens s.t. $\forall U \in \mathcal{U}'$ and $\forall u \in \mathcal{U}$, $k_U$ is derivable from $k_u$ iff $u \in U$ (C2 in Def. 1) */
for each $U_j \in \mathcal{U}'$, $|U_j| > 1$ do
  $cover_j := \{U_1, \ldots, U_n \subseteq \mathcal{U}' \mid \bigcup_{i=1}^{n} U_i = U_j\}$
  $\mathcal{T}' := \mathcal{T}' \cup \{t_{U_i,U_j} = k_{U_j} \oplus h(k_{U_i}, l_{U_j}) \mid U_i \in cover_j\}$
/* Step 2: define a key derivation structure */
generate a key $k_{\mathcal{S}}$ and a label $l_{\mathcal{S}}$ /* generate a key for the external server (C1 in Def. 2) */
$\mathcal{K} := \mathcal{K}' \cup \{k_{\mathcal{S}}\}$, $\mathcal{L} := \mathcal{L}' \cup \{l_{\mathcal{S}}\}$
$\mathcal{T} := \mathcal{T}'$
for each $U \in \mathcal{U}''$ do /* for each $U \in \mathcal{U}''$, compute $k_{U \cup \{\mathcal{S}\}}$ as the hash of $k_U$ (C1 in Def. 2) */
  $k_{U \cup \{\mathcal{S}\}} := h(k_U)$
  generate a label $l_{U \cup \{\mathcal{S}\}}$
  $\mathcal{K} := \mathcal{K} \cup \{k_{U \cup \{\mathcal{S}\}}\}$, $\mathcal{L} := \mathcal{L} \cup \{l_{U \cup \{\mathcal{S}\}}\}$
  $\mathcal{T} := \mathcal{T} \cup \{t_{\mathcal{S},U \cup \{\mathcal{S}\}} = k_{U \cup \{\mathcal{S}\}} \oplus h(k_{\mathcal{S}}, l_{U \cup \{\mathcal{S}\}})\}$ /* token from $k_{\mathcal{S}}$ to $k_{U \cup \{\mathcal{S}\}}$ (C2 in Def. 2) */
return($\langle \mathcal{K},\mathcal{L},\mathcal{T} \rangle$)
```

Fig. 2. Function that defines a key derivation structure

writers of the resource, we need to extend the set-based key derivation graph with the external server. The external server cannot however be treated as an authorized user of the system since it cannot access the plaintext of the outsourced resources. We then define a *key derivation structure* by extending the set-based key derivation graph to include also keys shared between authorized users and the server, which will be used to encrypt the write tags for enforcing write privileges (see Section 3.2). These additional keys can be derived both by the authorized users, applying a secure hash function to a key they already know (or can derive via a sequence of tokens), and by the external server, exploiting a token specifically added to the key derivation structure. Formally, a key derivation structure is defined as follows.

Definition 2 (Key derivation structure). *Let $\mathcal{U}$ be a set of users, $\mathcal{S}$ be an external server, $\mathcal{U}' \subseteq 2^{\mathcal{U}}$ be a family of subsets of users in $\mathcal{U}$ such that $\forall u \in \mathcal{U}$, $\{u\} \in \mathcal{U}'$, $\mathcal{U}''$ be a subset of $\mathcal{U}'$, and $\langle \mathcal{K}',\mathcal{L}',\mathcal{T}' \rangle$ be a set-based key derivation graph. A* key derivation structure *is a triple $\langle \mathcal{K},\mathcal{L},\mathcal{T} \rangle$, with $\mathcal{K}$ a set of keys, $\mathcal{L}$ the set of corresponding labels, and $\mathcal{T}$ a set of tokens, such that:*

1. $\mathcal{K} = \mathcal{K}' \cup \{k_{\mathcal{S}}\} \cup \{k_{U \cup \{\mathcal{S}\}} = h(k_U) \mid U \in \mathcal{U}''$ and h is a secure hash function$\}$;
2. $\mathcal{T} = \mathcal{T}' \cup \{t_{\mathcal{S},U \cup \{\mathcal{S}\}} \mid U \in \mathcal{U}''\}$.

Figure 2 illustrates function **Define_Key_Derivation_Structure** that builds a key derivation structure. The function receives as input a set $\mathcal{U}$ of users, an external server $\mathcal{S}$, and two families $\mathcal{U}'$ and $\mathcal{U}''$ of subsets of users in $\mathcal{U}$, with

$\mathcal{U}'' \subseteq \mathcal{U}'$. The function first defines a set-based key derivation graph $\langle\mathcal{K}',\mathcal{L}',\mathcal{T}'\rangle$ by leveraging on the algorithms in [8]. The function then extends the set-based key derivation graph by generating a key for the external server and for each set $U\cup\{\mathcal{S}\}$, with $U\in\mathcal{U}''$, and by inserting into $\mathcal{T}$ a token that allows the derivation of $k_{U\cup\{\mathcal{S}\}}$ from $k_{\mathcal{S}}$. The following theorem, whose proof is omitted for space constraints, formally shows that function **Define_Key_Derivation_Structure** correctly computes a key derivation structure.

Theorem 1 (Correctness). *Let $\mathcal{U}$ be a set of users, $\mathcal{S}$ be an external server, $\mathcal{U}' \subseteq 2^{\mathcal{U}}$ be a family of subsets of users in $\mathcal{U}$ such that $\forall u\in\mathcal{U}$, $\{u\}\in\mathcal{U}'$, and $\mathcal{U}''$ be subset of $\mathcal{U}'$. Triple $\langle\mathcal{K},\mathcal{L},\mathcal{T}\rangle$ computed by function* **Define_Key_Derivation_Structure** *in Figure 2 is a key derivation structure.*

Example 2. Consider a system with four users $\mathcal{U}=\{A,B,C,D\}$, a family $\mathcal{U}'=\{A,B,C,D,AB,CD,ABC,BCD\}$ of subsets of users, and a subset $\mathcal{U}''=\{C,AB,CD\}$ of $\mathcal{U}'$. Figure 3(c) illustrates the key derivation structure computed by function **Define_Key_Derivation_Structure** in Figure 2. In the figure, continuous lines correspond to tokens forming the set-based key derivation graph, dotted lines correspond to tokens added to define the key derivation structure, and double lines correspond to hash-based derivation.

3.2 Resource encryption, write tags, and access control enforcement

We consider the case of a data owner outsourcing her resources, which can be read or modified by other users. Each resource o is then associated with two access control lists: *i)* a read access list $r[o]$ reporting the set of users authorized to read o, and *ii)* a write access list $w[o]$ reporting the set of users authorized to write o. We assume the users authorized to write a resource also to be able to read it, that is, $\forall o \in\mathcal{O}$: $w[o]\subseteq r[o]$. As mentioned at the beginning of this section, we use write tags for enforcing write authorizations. For each resource $o\in\mathcal{O}$, the data owner defines a write tag, denoted $tag[o]$, by using a secure random function. The consideration of a secure random function ensures the independence of the write tag from the resource identifier (since otherwise readers not authorized to write could infer it) as well as from its content (since again readers could infer it or the server could infer information on the resource content). The write tag associated with a resource o is then encrypted with the key corresponding to the set U of users authorized to write it (i.e., $U = w[o]$) plus the server $\mathcal{S}$, that is, with key $k_{U\cup\{\mathcal{S}\}}$. Each resource $o \in \mathcal{O}$ is then stored at the external server in encrypted form together with the following metadata.

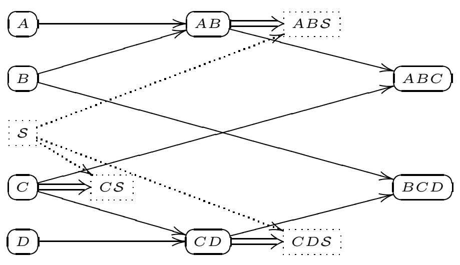
r_label: it is the label of the key with which the resource is encrypted, which is the key associated with the set of users authorized to read o (i.e., $l_{r[o]}$).
w_label: it is the label of the key shared by the users authorized to write o and the server $\mathcal{S}$ (i.e., $l_{w[o]\cup\{\mathcal{S}\}}$).
encw_tag: it is the write tag of the resource and is encrypted with the key identified by the label in w_label (i.e., $E(tag[o], k_{w[o]\cup\{\mathcal{S}\}})$).

o	r[o]	w[o]
o_1	ABC	AB
o_2	BCD	CD
o_3	ABC	C
o_4	AB	AB

(a)

r_label	w_label	o_id	encw_tag	encr_resource
l_{ABC}	l_{ABS}	1	α	zKZlJxVcCrC0g
l_{BCD}	l_{CDS}	2	β	t9qdJqC7AImXU
l_{ABC}	l_{CS}	3	γ	AxalPH8Kv37Ts
l_{AB}	l_{ABS}	4	δ	xwfPJSLn.MVqY

(b)



(c)

l_i	l_j	$t_{i,j}$
l_A	l_{AB}	$k_{AB} \oplus h(k_A, l_{AB})$
l_B	l_{AB}	$k_{AB} \oplus h(k_B, l_{AB})$
l_B	l_{BCD}	$k_{BCD} \oplus h(k_B, l_{BCD})$
l_C	l_{ABC}	$k_{ABC} \oplus h(k_C, l_{ABC})$
l_C	l_{CD}	$k_{CD} \oplus h(k_C, l_{CD})$
l_D	l_{CD}	$k_{CD} \oplus h(k_D, l_{CD})$
l_{AB}	l_{ABC}	$k_{ABC} \oplus h(k_{AB}, l_{ABC})$
l_{CD}	l_{BCD}	$k_{BCD} \oplus h(k_{CD}, l_{BCD})$
$l_{\mathcal{S}}$	l_{CS}	$k_{CS} \oplus h(k_{\mathcal{S}}, l_{CS})$
$l_{\mathcal{S}}$	l_{ABS}	$k_{ABS} \oplus h(k_{\mathcal{S}}, l_{ABS})$
$l_{\mathcal{S}}$	l_{CDS}	$k_{CDS} \oplus h(k_{\mathcal{S}}, l_{CDS})$

(d)

Fig. 3. An example of read and write acls (a), encrypted resources (b), key derivation structure (c), and tokens (d)

encr_resource: it is the encrypted version of the resource, encrypted with the key identified by the label in r_label (i.e., $E(o, k_{r[o]})$).

Let $\mathcal{U}$ be the set of users and $\mathcal{O}$ be the set of resources in the system, where each resource is associated with read and write access control lists as mentioned-above. To outsource her resources, the data owner must first compute keys and tokens forming the key derivation structure, by calling function **Define_Key_Derivation_Structure** in Figure 2. To this aim, she needs to define two families $\mathcal{U}'$ and $\mathcal{U}''$ of subsets of users in $\mathcal{U}$. $\mathcal{U}'$ corresponds to all the sets of users whose keys must be represented in the system for the correct enforcement of the authorizations. It then includes the singleton sets of users in $\mathcal{U}$, and the sets of users representing read and write access lists of resources in $\mathcal{O}$. $\mathcal{U}''$ is the subset of $\mathcal{U}'$ representing those sets of users that have to share a key with the external server. It then includes all the sets corresponding to the write access lists of resources in $\mathcal{O}$. The data owner then:

- communicates to each user u key k_u and to the external server key $k_{\mathcal{S}}$;
- computes and stores at the external server the encrypted resources and the associated metadata as described above;
- stores at the external server all tokens in $\mathcal{T}$ as a set of triples of the form $\langle l_i, l_j, t_{i,j} \rangle$ indicating that key with label l_j can be derived directly from key with label l_i through token $t_{i,j}$.

Example 3. Consider a system with four users $\mathcal{U}$=$\{A,B,C,D\}$ and four resources $\mathcal{O}$=$\{o_1,o_2,o_3,o_4\}$, and assume read and write acls of resources to be as in Figure 3(a) (read acls are the same as in Example 1). Figure 3(c) illustrates the key derivation structure computed as described in Example 2. Figure 3(b) and Figure 3(d) illustrate the encrypted resources and associated metadata, and the public tokens outsourced to the external server, respectively.

Enforcement of read authorizations is automatically guaranteed as in the original proposal [8]: the key with which a resource is encrypted can be known only to users authorized to read the resource. Enforcement of write authorizations can be easily delegated to the server that will accept a write operation on a resource only if the user shows knowledge of the corresponding (plaintext) write tag. Since the write tag of the resource is encrypted with $k_{w[o]\cup\{\mathcal{S}\}}$, besides the server only the authorized writers will be able to decrypt such an information. Also, since the server is assumed *honest-but-curious* (and therefore do not have interest to tamper with resources), this simple control allows us to enforce write authorizations. (More details in Section 4.)

It is easy to see that our approach guarantees: *i)* the correct *read authorization enforcement*, because the content of a resource is visible only to users authorized to read the resource; *ii)* the correct *write authorization enforcement*, because the write tag of a resource is visible only to users authorized to write the resource; *iii)* the *write control* by the server, because the tag of a resource is visible to the server. This is formalized by the following theorem, whose proof is omitted for space constraints.

Theorem 2 (Correct enforcement of authorizations). *Let $\mathcal{U}$ be a set of users, $\mathcal{S}$ be an external server, $\mathcal{O}$ be a set of resources with $r[o]$ and $w[o]$ the read and write access lists, respectively, of o. Our access control system satisfies the following conditions:*

1. *$\forall u \in \mathcal{U}$ and $\forall o \in \mathcal{O}$, u can decrypt encr_resource$[o]$ iff $u \in r[o]$* (read authorization enforcement)*;*
2. *$\forall u \in \mathcal{U}$ and $\forall o \in \mathcal{O}$, u can decrypt encw_tag$[o]$ iff $u \in w[o]$* (write authorization enforcement)*;*
3. *$\forall o \in \mathcal{O}$, $\mathcal{S}$ can decrypt encw_tag$[o]$* (write control)*.*

4 Write integrity control and resource management

According to our reference scenario our complicating factor in policy enforcement is to ensure proper protection of the confidentiality of data, while the server can be assumed trustworthy to manage resources and delegated actions. Nevertheless, it is important to provide a means to the data owner to verify that server and users are behaving properly. Providing such a control has a double advantage: *i)* it allows detecting resource tampering, due to the server not performing the required check on the write tags or directly tampering with resources, and *ii)* it discourages improper behavior by the server and by the users since they know that their improper behavior can be easily detected, and their updates recognized as invalid and discarded. In this section, we illustrate our approach for providing the data owner with a means to verify that modifications to a resource have been produced only by users authorized to write the resource. As discussed in the previous section, if the server performs the correct control on the write tags, such a property is automatically guaranteed. We therefore present how to perform a write integrity control to detect misbehavior (or laziness) by the server as well as misbehavior by users that can happen with the help of the server (not enforcing the control on the write tags since it is either colluding with the user or just behaving lazily) or without the help of the server (if the user improperly acquires the write tag for a resource by others).

A straightforward approach to provide such a write integrity control would be to apply a signature-based approach. This requires each user to have a pair ⟨private,public⟩ of keys and, when updating a resource, to sign the new resource content with her private key. The data owner can then check the write integrity by verifying that the signature associated with a resource correctly reflects the resource content and it is produced by a user authorized for the operation. Such an approach, while intuitive and simple, has however the main drawback of being computationally expensive (asymmetric encryption is considerably more expensive and therefore less efficient than symmetric encryption) and not well aligned with our approach, which - as a matter of fact - exploits symmetric encryption, tokens, and hash functions to provide efficiency in storage and processing. In the spirit of our approach, we then build our solution for controlling write integrity on HMAC functions [3].[4] In addition to the metadata illustrated in the previous section, we then associate with each resource three write integrity control fields:

encw_ts: it is the timestamp of the write operation, encrypted with the key $k_{w[o]\cup\{\mathcal{S}\}}$ corresponding to the group including the server and all the users in the write access list of o (i.e., $E(ts, k_{w[o]\cup\{\mathcal{S}\}})$);

user_tag: it is a HMAC computed with the key k_u of the user who performed the write operation over the resource concatenated with the user_tag of the resource prior to the write operation,[5] and the timestamp of the write operation (i.e., $\text{HMAC}(o||u\_t'||ts, k_u)$);

group_tag: it is a HMAC computed with the key $k_{w[o]}$ corresponding to the write access list of o over the resource concatenated with the timestamp of the write operation (i.e., $\text{HMAC}(o||ts, k_{w[o]})$).

Figure 4 summarizes the information associated with each encrypted resource.

At time zero, when the data owner outsources her resources to the server, the values of the user_tag and group_tag are those computed by the owner with her own key for the user_tag, and with the key of the write access list of the resource (to which the owner clearly belongs) for the group_tag. Every time a user updates a resource, it also updates its user_tag and group_tag.

[4] For common platforms, the ratio between the execution times of digital signatures and of HMAC is more than three orders of magnitude.

[5] The reason for including the user_tag of the resource prior to the write operation is to provide the data owner with a hash chain connecting all the resource versions (we assume the server to never overwrite resources but to maintain all their versions).

METADATA				RESOURCE	WRITE INTEGRITY CONTROL		
$l_{r[o]}$	$l_{w[o] \cup \{\mathcal{S}\}}$	o_id	$E(tag[o], k_{w[o] \cup \{\mathcal{S}\}})$	$E(o, k_{r[o]})$	$E(ts, k_{w[o] \cup \{\mathcal{S}\}})$	HMAC$(o \Vert u_t' \Vert ts, k_u)$	HMAC$(o \Vert ts, k_{w[o]})$
r_label	**w_label**	**<u>o_id</u>**	**encw_tag**	**encr_resource**	**encw_ts**	**user_tag**	**group_tag**

Fig. 4. Structure of outsourced resources

A user_tag is considered valid if it matches the resource content and it is produced by a user in the write access list of the resource. The user_tag provides write integrity (meaning the resource has been written by an authorized user) and accountability of user actions. In fact, since the data owner knows the key k_u of every user u (which she generated and distributed), she can check the validity of the user_tag and detect possible mismatches, corresponding to unauthorized writes. In addition, every write operation considered valid (according to the control on the user_tag) cannot be repudiated by the user u whose key k_u has generated the HMAC. The consideration of group_tag extends the ability of checking the validity of the write operations (i.e., write integrity) also to all the users in the write access list of the resource. To guarantee the integrity of the metadata associated with each resource and that no resource is dropped from the system, we adopt traditional approaches based on aggregate signatures [13] (i.e., the first four fields in Figure 4 are signed as a whole by the data owner, providing efficient and scalable integrity check for the structure with one signature).

While we assume the server to be trustworthy and therefore not interested in tampering with the resources, we note that the user_tag would allow also to detect possible tampering of the server with the resource (since not being an authorized writer, the server will not be able to produce a valid user_tag). The server could also tamper with the write authorizations, by decrypting the write tag and encrypting it with the key corresponding to a different write access list. However, the improper inclusion of a user in the write access list does not have any different effect than when the server does not perform the control, since the user improperly included in the write access list will not be able to produce a valid user_tag. Analogously, the improper removal of a user from the write access list has the same effects as when the server refuses its services.

Unauthorized write operations in the case of a well behaving server can only happen if a user has improperly acquired or received from other authorized users the write tag of a resource. Whichever the case, the user will be able to provide neither a valid user_tag nor a valid group_tag for the resource. Also, the data owner and any user authorized to write the resource will be able to detect the invalidity of the group_tag, since the key used to compute the HMAC will not correspond to the key of $w[o]$.

5 Related work

In the last few years, several research efforts have been devoted to enable data owners to outsource the storage and management of their data to possibly nonfully trusted third parties [16]. Most proposals have addressed the problem of efficiently performing queries on outsourced encrypted data, without decrypting sensitive information at the server side (e.g., [1, 5, 7, 10, 17]), or the problem of protecting data integrity and authenticity (e.g., [11, 13, 14, 18]). The problem of enforcing an authorization policy on outsourced data is orthogonal to these issues and has recently received the attention of the research community (e.g., [8, 12, 19]). The proposal in [12] protects access to XML documents by encrypting different portions of the XML tree using different keys. The solution in [8] combines selective encryption and key derivation strategies for controlling accesses to outsourced resources. This approach also permits to delegate to the server the management of policy updates. More recently, attribute-based encryption has been proposed for providing system scalability [19].

All the approaches above consider read access privileges only and few works have addressed the issue of enforcing write privileges. Raykova et al. [15] adopt selective encryption to enforce read and write privileges on outsourced data. The authors introduce a two-layer access control model. The server restricts read/write accesses at the level of block. To enforce the authorization policy at the granularity of resources within blocks, the proposed system adopts asymmetric encryption and defines two key derivation hierarchies: one for private keys (to enforce read privileges) and one for public keys (to enforce write privileges). This solution cannot easily enforce policies where resources with the same read access control policy can be modified by two different sets of users. In this case, the approach in [15] can be adapted by associating different pairs of keys to the same group of users, thus increasing key management overhead. Zhao et al. [20] propose attribute-based encryption and attribute-based signature techniques to enforce read and write access privileges, respectively. This approach requires the presence of a trusted party for correct policy enforcement. Also, attribute-based techniques are computationally more expensive than traditional symmetric encryption.

6 Conclusions

In this paper, we presented an approach for supporting both read and write privileges on outsourced encrypted data. The proposed solution relies on the use of symmetric encryption, hashing, and HMAC functions for enforcing access control in an efficient and effective way. Our proposal performs then a step towards the development of solutions actually applicable to real-world scenarios where efficiency and scalability are mandatory.

Acknowledgements. This work was partially supported by the EC within the 7FP, under grant agreement 257129 (PoSecCo), by the Italian Ministry of Research within the PRIN 2008 project "PEPPER" (2008SY2PH4), and by the Università degli Studi di Milano within the project "PREVIOUS". The work of Sushil Jajodia was partially supported by the National Science Foundation under grants CCF-1037987 and CT- 20013A.

References

1. Agrawal, R., Kierman, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proc. of SIGMOD 2004. Paris, France (June 2004)
2. Atallah, M., Frikken, K., Blanton, M.: Dynamic and efficient key management for access hierarchies. In: Proc. of CCS 2005. Alexandria, VA, USA (November 2005)
3. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Proc. of CRYPTO 1996. Santa Barbara, CA, USA (August 1996)
4. Cimato, S., Gamassi, M., Piuri, V., Sassi, R., Scotti, F.: Privacy-aware biometrics: Design and implementation of a multimodal verification system. In: Proc. of ACSAC 2008. Anaheim, CA, USA (December 2008)
5. Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational DBMSs. In: Proc. of CCS 2003. Washington, DC, USA (October 2003)
6. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P.: Fine grained access control for SOAP e-services. In: Proc. of WWW 2001. Hong Kong, China (May 2001)
7. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: A data outsourcing architecture combining cryptography and access control. In: Proc. of CSAW 2007. Fairfax, VA, USA (November 2007)
8. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Encryption policies for regulating access to outsourced data. ACM TODS 35(2), 12:1–12:46 (April 2010)
9. Gamassi, M., Piuri, V., Sana, D., Scotti, F.: Robust fingerprint detection for access control. In: Proc. of RoboCare 2005. Rome, Italy (May 2005)
10. Hacigümüs, H., Iyer, B., Mehrotra, S., Li, C.: Executing SQL over encrypted data in the database-service-provider model. In: Proc. of the SIGMOD 2002. Madison, WI, USA (June 2002)
11. Merkle, R.: A certified digital signature. In: Proc. of CRYPTO 1989. Santa Barbara, CA, USA (August 1989)
12. Miklau, G., Suciu, D.: Controlling access to published data using cryptography. In: Proc. of VLDB 2003. Berlin, Germany (September 2003)
13. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. ACM TOS 2(2), 107–138 (May 2006)
14. Pang, H., Jain, A., Ramamritham, K., Tan, K.: Verifying completeness of relational query results in data publishing. In: Proc. of SIGMOD 2005. Baltimore, MA, USA (June 2005)
15. Raykova, M., Zhao, H., Bellovin, S.: Privacy enhanced access control for outsourced data sharing. In: Proc. of FC 2012. Bonaire (February-March 2012)
16. Samarati, P., De Capitani di Vimercati, S.: Data protection in outsourcing scenarios: Issues and directions. In: Proc. of ASIACCS 2010. China (April 2010)
17. Wang, H., Lakshmanan, L.V.S.: Efficient secure query evaluation over encrypted XML databases. In: Proc. of VLDB 2006. Seoul, Korea (September 2006)
18. Xie, M., Wang, H., Yin, J., Meng, X.: Integrity auditing of outsourced data. In: Proc. of VLDB 2007. Vienna, Austria (September 2007)
19. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: Proc. of INFOCOM 2010. San Diego, CA, USA (March 2010)
20. Zhao, F., Nishide, T., Sakurai, K.: Realizing fine-grained and flexible access control to outsourced data with attribute-based cryptosystems. In: Proc. of ISPEC 2011. Guangzhou, China (May 2011)