Over-encryption: Management of Access Control Evolution on Outsourced Data

Sabrina De Capitani di Vimercati
DTI - Università di Milano
26013 Crema - Italy
decapita@dti.unimi.it

Sara Foresti
DTI - Università di Milano
26013 Crema - Italy
foresti@dti.unimi.it

Sushil Jajodia
CSIS - George Mason University
Fairfax, VA 22030-4444
jajodia@gmu.edu

Stefano Paraboschi
DIIMM - Università di Bergamo
24044 Dalmine - Italy
parabosc@unibg.it

Pierangela Samarati
DTI - Università di Milano
26013 Crema - Italy
samarati@dti.unimi.it

ABSTRACT

Data outsourcing is emerging today as a successful paradigm allowing users and organizations to exploit external services for the distribution of resources. A crucial problem to be addressed in this context concerns the enforcement of selective authorization policies and the support of policy updates in dynamic scenarios.

In this paper, we present a novel solution to the enforcement of access control and the management of its evolution. Our proposal is based on the application of selective encryption as a means to enforce authorizations. Two layers of encryption are imposed on data: the inner layer is imposed by the owner for providing initial protection, the outer layer is imposed by the server to reflect policy modifications. The combination of the two layers provides an efficient and robust solution. The paper presents a model, an algorithm for the management of the two layers, and an analysis to identify and therefore counteract possible information exposure risks.

1. INTRODUCTION

Contrary to the vision of a few years ago, where many predicted that Internet users would have in a short time exploited the availability of pervasive high-bandwidth network connections to activate their own servers, users are today, with increasing frequency, resorting to service providers for disseminating and sharing resources they want to make available to others.

The continuous growth of the amount of digital information to be stored and widely distributed, together with the always increasing storage, support the view that service providers will be more and more requested to be responsible for the storage and the efficient and reliable distribution of content produced by others, realizing a "data outsourcing" architecture on a wide scale. This important trend is particularly clear when we look at the success of services like YouTube, Flickr, Blogger, MySpace, and many others in the "social networking" environment.

When storage and distribution do not involve publicly releasable resources, selective access techniques must be enforced. In this context, it is legitimate for the data owner to demand the data not to be disclosed to the service provider itself, which, while trustworthy to properly carry out the resource distribution functions, should not be allowed access to the resource content.

The problem of outsourcing resource management to a "honest but curious" service has recently received considerable attention by the research community and several advancements have been proposed. The different proposals require the owner to encrypt the data before outsourcing them to the service. Most proposals assume that the data are encrypted with a single key only [7, 10, 11]. In such a context, either authorized users are assumed to have the complete view on the data or, if different views need to be provided to different users, the data owner needs to participate in the query execution to possibly filter the result of the service provider. Other proposals [13, 18], developed for the protection of XML documents, avoid this problem by applying selective encryption, where different keys are used to encrypt different portions of the XML tree. In the outsourcing scenario, all these proposals, in case of updates of the authorization policy, would require to re-encrypt the resources and resend them to the service. If the owner does not have the resources stored locally, a further preliminary step is needed to re-acquire them from the service and decrypt them.

In this paper, we propose a solution that removes these issues, facilitating the successful development of outsourcing data in emerging scenarios. In particular, we look at the problem of defining and assigning keys to users, by exploiting hierarchical key assignment schemes [3, 4, 8, 15], and of efficiently supporting policy changes. Indeed, in scenarios involving potentially huge sets of resources of considerable size, re-encryption and re-transmission by the owner may not be acceptable. The advantage compared with a solution requiring to re-send a novel encrypted version of the resource

VLDB '07, September 23-28, 2007, Vienna, Austria.

is typically huge and arbitrarily large (if the resource has a size of 1 GByte and the request to the server requires a 100-byte packet, in terms of network traffic, compared to the transmission of the re-encrypted resource, the improvement is in the order of 10^7).

The contribution of this paper is threefold. First (Section 2), we propose a formal base model for the correct application of selective encryption. The model allows the definition of an encryption policy equivalent to the authorization policy to be enforced on the resources. We note that, while it is in principle advisable to leave authorization-based access control and cryptographic protection separate, in the outsourcing scenario such a combination can prove successful: selective encryption allows selective access to be enforced by the service provider itself without the owner intervention.

Second (Section 3 and Section 4), building on the base model we propose the use of a two-layer approach to enforce selective encryption without requesting the owner to re-encrypt the resources every time there is a change in the authorization policy. The first layer of encryption is applied by the data owner at initialization time (when releasing the resource for outsourcing), the second layer of encryption is applied by the service itself to take care of dynamic policy changes. Intuitively, the two-layer encryption allows the owner to outsource, besides the resource storage and dissemination, the authorization policy management, while not releasing data to the provider.

Third (Section 5), we provide a characterization of the different views of the resources by different users and characterize potential risks of information exposures due to dynamic policy changes. The investigation allows us to conclude that, while an exposure risk may exist, it is well defined and identifiable. This allows the owner to address the problem and minimize it at design time. Also, the fact that exposure arises only in specific situations and over well identified resources, allows the owner to completely eliminate it by resorting to re-encryption when necessary.

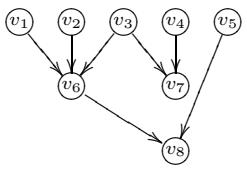
An important strength of our solution is that it does not substitute the current proposals, rather it complements them, enabling them to support encryption in a selective form and easily enforce dynamic policy changes.

2. BASE MODEL

Consistently with the typical outsourcing scenario, we distinguish three roles: the *data owner*, who creates the resource; the *server*, who receives by the data owner a resource in an encrypted format and makes it available on the network; and the *user*, who exploits the knowledge of a small shared secret (a key) and possibly additional public information (typically available on the server) to access the resource plaintext.

2.1 Keys and tokens

We assume the existence of a set of *symmetric* encryption keys $\mathcal{K}$ in the system. Also, we exploit the existence of the key derivation method proposed by Atallah's et al. [4] based on the definition and computation of *public tokens* that allow the derivation of keys from other keys. Given two keys k_i and k_j, a token $t_{i,j}$ is defined as $t_{i,j} = k_j \oplus h(k_i, l_j)$, where l_j is a publicly available label associated with k_j, $\oplus$ is the bit-a-bit `xor` operator and h is a deterministic cryptographic function (in [4] it was proposed the use of a secure hash function). Since keys need to remain secret, while tokens are public, the use of tokens greatly simplifies key management. Key derivation via tokens can be applied in chains: a chain of tokens is a sequence $t_{i,l}, \ldots, t_{n,j}$ of tokens such that $t_{c,d}$ follows $t_{a,b}$ in the chain only if $b = c$. This is formally captured by the following definition, associating with each key the set of keys reachable by it via chains of tokens.



(a)

source	dest	value
$v_1.k$	$v_6.k$	$v_6.k \oplus h(v_1.k, l_6)$
$v_2.k$	$v_6.k$	$v_6.k \oplus h(v_2.k, l_6)$
$v_3.k$	$v_6.k$	$v_6.k \oplus h(v_3.k, l_6)$
$v_3.k$	$v_7.k$	$v_7.k \oplus h(v_3.k, l_7)$
$v_4.k$	$v_7.k$	$v_7.k \oplus h(v_4.k, l_7)$
$v_5.k$	$v_8.k$	$v_8.k \oplus h(v_5.k, l_8)$
$v_6.k$	$v_8.k$	$v_8.k \oplus h(v_6.k, l_8)$

(b)

Figure 1: An example of key derivation

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
A	0	0	0	0	1	1	1	1
B	0	0	0	0	1	1	1	1
C	1	1	1	1	1	1	1	1
D	0	0	1	1	0	0	0	0
E	0	0	0	0	0	0	0	1

Figure 2: An example of access matrix

u	$\phi(u)$
A	$v_1.k$
B	$v_2.k$
C	$v_3.k$
D	$v_4.k$
E	$v_5.k$

(a)

r	$\phi(r)$
r_1,r_2	$v_3.k$
r_3,r_4	$v_7.k$
r_5,r_6,r_7	$v_6.k$
r_8	$v_8.k$

(b)

Figure 3: An example of key assignment

Definition 1. (*Key derivation function*).
Let $\mathcal{K}$ be the set of symmetric encryption keys in the system, and $\mathcal{T}$ the set of tokens in the public catalog.
The *direct key derivation function* $\tau : \mathcal{K} \mapsto 2^{\mathcal{K}}$ is defined as $\tau(k_i)=\{k_j \in \mathcal{K} \mid \exists t_{i,j} \in \mathcal{T}\}$.
The *key derivation function* $\tau^* : \mathcal{K} \mapsto 2^{\mathcal{K}}$ is a function such that $\tau^*(k_i)$ is the set of keys derivable from k_i by chains of tokens, including the key itself (chain of length 0).

We can graphically represent keys and tokens through a graph, having a vertex v_i associated with each key in the system, denoted $v_i.k$, and an edge connecting two vertices (v_i, v_j) if token $t_{i,j}$ belongs to the public token catalog $\mathcal{T}$. Chains of tokens then correspond to paths in the graph. In other words, the key derivation function associates with each $\tau^*(k)$ the keys of vertices reachable from the vertex associated with k in the graph. For instance, Figure 1(a) represents an example of graph corresponding to a set of 8 keys, along with its public token catalog composed of 7 items (see Figure 1(b)). As an example, $\tau(v_3.k)=\{v_6.k, v_7.k\}$ and $\tau^*(v_3.k)=\{v_3.k, v_6.k, v_7.k, v_8.k\}$.

2.2 Access control and encryption policies

Consistently with the data distribution and dissemination scenario, we assume access by users to the outsourced data to be read-only. The data owner therefore defines a *discretionary access control policy* to regulate read operations on the outsourced resources. Authorizations are of the form $\langle user, resource \rangle$.[1]

[1]For the sake of simplicity, here we do not deal with the fact

Let $\mathcal{U}$ be the set of users of the system and $\mathcal{R}$ the set of outsourced resources. Authorizations can be modeled via a traditional *access matrix* $\mathcal{A}$, with a row for each user $u \in \mathcal{U}$, a column for each resource $r \in \mathcal{R}$. Each entry $\mathcal{A}[u, r]$ is set to 1 if u can access r; 0 otherwise. Figure 2 illustrates an example of access matrix with five users (A, B, C, D, E) and eight resources (r_1, r_2, ..., r_8). Given an access matrix $\mathcal{A}$ over sets $\mathcal{U}$ and $\mathcal{R}$, $acl(r)$ denotes the *access control list* of r (i.e., the set of users that can access r), while $cap(u)$ denotes the *capability list* of u (i.e., the set of resources that u can access). For instance, with reference to the matrix in Figure 2, $acl(r_1)=\{C\}$ and $cap(B)=\{r_5,r_6,r_7,r_8\}$.

Our initial goal is to represent the authorizations by means of proper encryption and key distributions. To avoid users having to store and manage a huge number of (secret) keys, we exploit the key derivation method via tokens. Exploiting tokens, the release to each user of a set of keys $K = \{k_1, \ldots, k_n\}$ can be equivalently obtained by the release to each user of a single key $k_i \in K$ and the publication of a set of tokens allowing to derive (directly or indirectly) all keys $k_j \in K, j \neq i$. A major advantage of using tokens is that they are public (typically they will be stored together with resources) and allow each user to derive multiple encryption keys, while having to securely manage a single one. In the following, we use $\mathcal{K}$ and $\mathcal{T}$ to denote the set of keys and tokens defined in the system, respectively.

We assume that each user is associated with a single key, communicated to her by the owner on a secure channel at the time the user joins the system. Also, each resource can be encrypted by using a single key.

Definition 2. (*Key assignment*). A *key assignment* is a function $\phi : \mathcal{U} \cup \mathcal{R} \mapsto \mathcal{K}$, which associates with each user $u \in \mathcal{U}$ the (single) key released to her and with each resource $r \in \mathcal{R}$ the (single) key with which the resource is encrypted.

It is easy to see that, by Definition 1, each user u can retrieve (via her own key $\phi(u)$ and the set of public tokens $\mathcal{T}$) all the keys derivable from $\phi(u)$, that is, all the keys in $\tau^*(\phi(u))$. In the following, we use $\phi^*(u)$ as a short hand for $\tau^*(\phi(u))$.
Figure 3 represents an example of key assignment, with reference to the key graph in Figure 1(a). Composing the graph in Figure 1(a) with the function defined in Figure 3(a), we obtain the set of keys each user knows. As an example, $\phi^*(B)=\tau^*(v_2.k)=\{v_2.k,v_6.k,v_8.k\}$. A key derivation defined by tokens and the corresponding key assignment realize an encryption policy. This is captured by the following definition.

Definition 3. (*Encryption policy*). Let $\mathcal{E}=(\tau^*, \phi)$ be an *encryption policy* composed of a key derivation function τ^* and a key assignment function ϕ.

An encryption policy $\mathcal{E}$ is said to *correctly model* a set of authorizations $\mathcal{A}$ if and only if users are able to decrypt all and only the resources for which they have the authorization. This is formally captured by the following definition.

Definition 4. (*Correctness*). Let $\mathcal{E}=(\tau^*, \phi)$ be an encryption policy, and $\mathcal{A}$ an access control policy. $\mathcal{E}$ is said to *correctly enforce* $\mathcal{A}$, denoted $\mathcal{E} \Longleftrightarrow \mathcal{A}$, iff both the following conditions hold:

- *Soundness.* $\forall u \in \mathcal{U}, r \in \mathcal{R}$: $\phi(r) \in \phi^*(u) \Longrightarrow \mathcal{A}[u, r] = 1$
- *Completeness.* $\forall u \in \mathcal{U}, r \in \mathcal{R}$: $\mathcal{A}[u, r] = 1 \Longrightarrow \phi(r) \in \phi^*(u)$

As an example, the encryption policy illustrated in Figures 1 and 3 correctly enforces the policy represented by the access matrix in Figure 2.

A straightforward approach for defining a correct $\mathcal{E}$ can exploit the hierarchy among sets of users, which is induced by the partial ordering of sets of users, based on the set containment relationship ($\subseteq$) [9]. This strategy works as follows:

- define a key for each access control list as well as for each singleton set of users (when not already included in the *acl*s);
- define tokens between keys corresponding to sets of users in hierarchical relationships; [2]
- assign to each user u the key corresponding to its singleton group $\{u\}$;
- encrypt each resource r with the key associated with $acl(r)$.

As an example, consider policy $\mathcal{A}$ in Figure 2. Since there is a set of 4 different *acls* (i.e., $\{C\}$, $\{C, D\}$, $\{A, B, C\}$, $\{A, B, C, E\}$), which includes a singleton *acl*, and 5 users, we need to define 8 different keys. In particular, keys k_1,... k_5 are associated, in the order, with users A,...,E, k_6 is associated with $\{A, B, C\}$, k_7 with $\{C, D\}$, and k_8 with $\{A, B, C, E\}$. We then define a token between each pair of keys $(k_i, k_j), i, j = 1 \ldots, 8$ and $i \neq j$, such that the set of users corresponding to k_i is included in the set of users corresponding to k_j. Figure 1(a) illustrates a graphical representation of these keys and tokens, where vertex v_i corresponds to key k_i. Resources $r_1, \ldots, r_8$ are then encrypted as shown in Figure 3(b).

3. TWO LAYER MODEL

The model described in Section 2 assumes keys and tokens are computed, on the basis of the existing authorization policy, prior to sending the encrypted resources to the server. While we can note that, in general, the authorizations specified at initialization time may not change frequently, many situations require dynamic changes of authorizations, for example, for granting privileges to new users.

Every time an authorization on a resource r is granted or revoked, $acl(r)$ changes accordingly. In terms of the encryption policy this implies the need to change the key used to encrypt the resource (i.e., $\phi(r)$) to make it accessible (i.e., intelligible) only to users in the new *acl*. This operation requires then decrypting the resource (with the key with which it is currently encrypted) to retrieve the original plaintext (the owner may possibly store resources only on the external provider without keeping a local copy), and then re-encrypt it with the new key. Imposing such an overhead (in terms

that authorizations can be specified for predefined groups of users and groups of resources. Our approach will support dynamic grouping, therefore subsuming any statically defined group.

[2] As usual, only direct relationships are considered [12].

of both transmission and computation) for managing authorization changes does not fit our working assumption, where the owner outsources its data since it does not have the necessary resources and transmission channels to manage them.

We put forward the idea of outsourcing to the server, besides the resource storage, the authorization management as well. Note that this delegation is possible since the server is considered trustworthy to properly carry out the service. Recall, however, that the server is not trusted with confidentiality (honest but curious). For this reason, our solution has been thought of with the goal of minimizing the case of the server colluding with the users to breach data confidentiality (see Section 5). The way to make this possible is via a solution that enforces policy changes on encrypted resources themselves (without the need of decrypting them), and can then be performed by the server.

3.1 Two-layer encryption

To delegate policy changes enforcement to the server, avoiding re-encryption for the data owner, we adopt a two layer encryption approach. The owner encrypts the resources and sends them to the server in encrypted form; the server can impose another layer of encryption (following directions by the data owner).

We then distinguish two layers of encryption.

- **Base Encryption Layer** (BEL), performed by the data owner before transmitting data to the server. It enforces encryption on the resources according to the policy existing at initialization time.
- **Surface Encryption Layer** (SEL), performed by the server over the resources already encrypted by the data owner. It enforces the dynamic changes over the policy.

Both layers enforce encryption by means of a set of symmetric keys and a set of public tokens between these keys, as illustrated in Section 2, although some adaptations are necessary as explained respectively in Sections 3.1.1 and 3.1.2.

In terms of efficiency, the use of a double layer of encryption does not appear as a significant computational burden; experience shows that current systems have no significant delay when managing encryption on data coming from either the network or local disks, this is also testified by the widespread use of encryption on network traffic and for protecting the storage of data on local file systems [16].

3.1.1 Base Encryption Layer

Compared with the model presented in Section 2, in the BEL level we distinguish two kinds of keys: *derivation keys* and *access keys*. Access keys are the ones actually used to encrypt resources, while derivation keys are used to provide the derivation capability via tokens, i.e., tokens can be defined only with the derivation key as starting point. Each derivation key k is always associated with an access key k_a obtained by applying a secure hash function to k, that is, $k_\mathsf{a} = h(k)$. In other words, keys at the BEL level always go in pairs $\langle k, k_\mathsf{a}\rangle$. The rationale for this evolution is to distinguish the two roles associated with keys, namely: enabling key derivation (applying the corresponding tokens) and enabling resource access. The reason for which such a distinction is needed will be clear in Section 4.

A key derivation function $\tau^*_\mathsf{b} : \mathcal{K} \mapsto 2^\mathcal{K}$ associates with each derivation key the set of keys derivable from it - either directly or indirectly - via the public token catalog and/or the application of the hashing function. Again, the key derivation relationship is represented by means of a graph, which now has a vertex b for each pair of keys $\langle k, k_\mathsf{a}\rangle$ and an edge connecting vertices (b_i, b_j) if there is a token in the public catalog allowing the derivation of either $b_j.k$ or $b_j.k_\mathsf{a}$ from $b_i.k$. To graphically distinguish tokens leading to derivation keys from tokens leading to access keys we use dashed lines for the latter. Given a key k_i, $\tau^*_\mathsf{b}(k_i)$ returns all keys reachable from vertex b with $b.k{=}k_i$ by following paths of tokens and hashing. Note that dashed edges can only appear as the last step of the path (and they allow the derivation of the access key only).

A *key assignment* is a function $\phi_\mathsf{b} : \mathcal{U} \cup \mathcal{R} \mapsto \mathcal{K}$ that associates with each user $u{\in}\,\mathcal{U}$ the (single) *derivation* key released to the user by the data owner and with each resource $r{\in}\,\mathcal{R}$ the (single) *access* key with which the resource is encrypted by the data owner. Figure 4 illustrates an example of BEL corresponding to the example introduced in Section 2.

3.1.2 Surface Encryption Layer

As in the base model of Section 2, at the SEL level there is no distinction between derivation and access keys (intuitively a single key carries out both functions).

Again, we define a key derivation function $\tau^*_\mathsf{s} : \mathcal{K} \mapsto 2^\mathcal{K}$ that can be represented by means of a graph having a vertex for each key k defined at SEL and an edge connecting vertices (s_i, s_j) if there is a token in the public catalog allowing the derivation of $s_j.k$ from $s_i.k$.

A *key assignment* is a function $\phi_\mathsf{s} : \mathcal{U} \cup \mathcal{R} \mapsto \mathcal{K} \cup \{\text{NULL}\}$ that associates with each user $u{\in}\,\mathcal{U}$ the (single) key released to the user by the server and with each resource $r{\in}\,\mathcal{R}$ the (single) key with which the resource is encrypted by the server (if any).

3.1.3 BEL *and* SEL *combination*

In the two-layer approach, each resource can then be encrypted twice: at the BEL level first, and at the SEL level then. Users can access resources only via the SEL level. Each user u receives two keys: one to access the BEL and the other to access the SEL.[3] Users will be able to access resources for which they know both the keys (BEL and SEL) used for encryption.

The consideration of the two levels requires to restate the correctness property of the encryption policy, which is now defined as follows.

Definition 5. (*Correctness*). Let $\mathcal{E}_\mathsf{b}{=}(\tau^*_\mathsf{b}, \phi_\mathsf{b})$ be an encryption policy at the BEL level, $\mathcal{E}_\mathsf{s}{=}(\tau^*_\mathsf{s}, \phi_\mathsf{s})$ be an encryption policy at the SEL level, and $\mathcal{A}$ an access control policy. The pair $\langle\mathcal{E}_\mathsf{b},\mathcal{E}_\mathsf{s}\rangle$ is said to *correctly enforce* $\mathcal{A}$, denoted $\langle\mathcal{E}_\mathsf{b},\mathcal{E}_\mathsf{s}\rangle \Longleftrightarrow \mathcal{A}$, iff both the following conditions hold:

- *Soundness.* $\forall u \in \mathcal{U}, r \in \mathcal{R}$:
  $\phi_\mathsf{b}(r) \in \phi^*_\mathsf{b}(u) \wedge \phi_\mathsf{s}(r) \in \phi^*_\mathsf{s}(u) \Longrightarrow \mathcal{A}[u, r] = 1$
- *Completeness.* $\forall u \in \mathcal{U}, r \in \mathcal{R}$:
  $\mathcal{A}[u, r] = 1 \Longrightarrow \phi_\mathsf{b}(r) \in \phi^*_\mathsf{b}(u) \wedge \phi_\mathsf{s}(r) \in \phi^*_\mathsf{s}(u)$

[3]To simplify key management, the user key $\phi_\mathsf{s}(u)$ for SEL can be obtained by the application of a secure hash function from the user key $\phi_\mathsf{b}(u)$ for BEL. In this case, the data owner needs to send in the initialization phase to the server the list of SEL keys of each user.

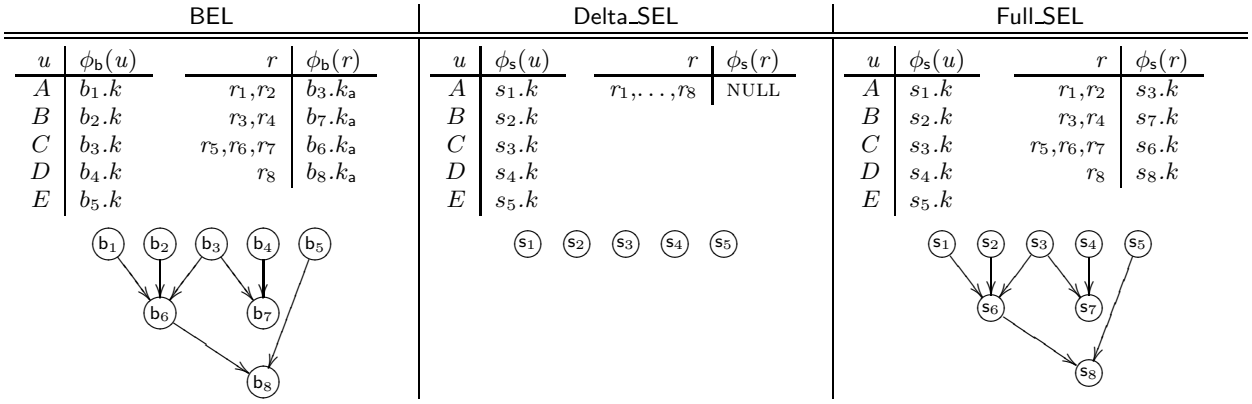


Figure 4: An example of BEL and SEL combination with the Delta_SEL and the Full_SEL approaches

In principle, any encryption policy at BEL and SEL can be specified as long as their combination is correct with respect to the authorizations.

Let $\mathcal{A}$ be the authorization policy at the initialization time and let $(\tau_{\mathsf{b}}^*, \phi_{\mathsf{b}})$ be the encryption policy at the BEL level correctly enforcing it (i.e., $(\tau_{\mathsf{b}}^*, \phi_{\mathsf{b}}) \iff \mathcal{A}$). We envision two basic approaches that can be followed in the construction of the two levels.

Full_SEL. The SEL policy is initialized to reflect exactly (i.e., to repeat) the BEL policy: for each derivation key in BEL a corresponding key is defined in SEL; for each token in BEL, a corresponding token is defined in SEL. Note that the key derivation function τ_{s}^* corresponds to a graph which is isomorphic to the one existing at the BEL level. Each user is assigned the unique key $\phi_{\mathsf{s}}(u)$ corresponding to her $\phi_{\mathsf{b}}(u)$. Each resource is encrypted with the unique key $\phi_{\mathsf{s}}(r)$ corresponding to the unique derivation key associated with access key $\phi_{\mathsf{b}}(r)$. The SEL policy models exactly the BEL policy, and hence by definition is correct with respect to the access control policy (i.e., $(\tau_{\mathsf{s}}^*, \phi_{\mathsf{s}}) \iff \mathcal{A}$).

Delta_SEL. The SEL policy is initialized to not carry out any over-encryption. Each user u is assigned a unique key $\phi_{\mathsf{s}}(u)$. No encryption is performed on resources, that is, $\forall r \in \mathcal{R} : \phi_{\mathsf{s}}(r) = \text{NULL}$. The SEL level itself does not provide any additional protection at start time, but it does not modify the accesses allowed by BEL.

We note that a third approach could be possible, where the authorization enforcement is completely delegated at the SEL level and the BEL simply applies a uniform over-encryption (i.e., with the same key released to all users) to protect the plaintext content from the server's eyes. We do not consider this approach as it presents a significant exposure to collusion (Section 5).

It is easy to see that all the approaches described produce a correct two layer encryption. In other words, given a correct encryption policy at the BEL level, the approaches produce a SEL level such that the pair $\langle \mathcal{E}_{\mathsf{b}}, \mathcal{E}_{\mathsf{s}} \rangle$ correctly enforces the authorizations.

The reason for considering both the Full_SEL and Delta_SEL approaches is the different performance and protection guarantees that they enjoy. In particular, Full_SEL always requires double encryption to be enforced (even when authorizations remain unvaried), thus doubling the decryption load of users for each access. By contrast, the Delta_SEL approach requires double encryption only when actually needed to enforce a change in the authorizations. However, as we will see in Section 5, the Delta_SEL is characterized by greater information exposure, which instead does not affect the Full_SEL approach. The choice between one or the other can then be a trade-off between costs and resilience to attacks.

We close this section with a remark on the implementation. In the illustration of our approach we always assume over-encryption to be managed with a direct and complete encryption and decryption of the resource, as needed. We note however that the server can, at the SEL level, apply a *lazy encryption* approach, similar to the *copy-on-write* (COW) strategy used by most operating systems, and actually over-encrypt the resource when it is first accessed (and then storing the computed encrypted representation); the server may choose also to always store the BEL representation and then dynamically apply the encryption driven by the SEL when users access the resource.

4. POLICY UPDATES

Policy update operations can be of three kinds: *1)* inserting/deleting a user; *2)* inserting/deleting a resource; and *3)* granting/revoking an authorization. We note that insertion/deletion of users and resources do not have any impact on the policy (and consequently on the keys) as long as these users and resources are not involved in authorizations. Without loss of generality, we restrict ourselves to the consideration of granting and revoking of authorizations (with the note that deleting a user/resource implies revoking all the authorizations in which the user/resource is involved). Also, for the sake of simplicity, we assume grant and revoke operations to be referred to a single user u and a single resource r. Their extension to sets of users and resources is immediate.

Policy updates are demanded and regulated by the owner (i.e., at the BEL level). The two-layer approach enables the enforcement of policy updates without the need for the owner to re-encrypt, and to resend them to the server. By contrast, the owner just adds (if necessary) some tokens at the BEL level and delegates policy changes to the SEL level by possibly requesting the server to over-encrypt the resources. The SEL level (enacted by the server) receives

over-encryption requests by the BEL level (under the control of the data owner) and operates accordingly, adjusting tokens and possibly encrypting (and/or decrypting) resources.

Before analyzing grant and revoke operations let us then see the working of over-encryption at the SEL level.

4.1 Over-encrypt

The SEL level regulates the update process through over-encryption of resources. It receives from the BEL requests of the form **over-encrypt**(R, U) corresponding to the demand to the SEL to make the set of resources R accessible only to users in U. Note here that the semantics is different in the two different encryption modes. In the Full_SEL approach, over-encryption must reflect the actual access control policy existing at any given time. In other words, it must reflect, besides the - dynamic - policy changes not reflected in the BEL, also the BEL policy itself. In the Delta_SEL approach, over-encryption is demanded only when additional restrictions (with respect to those enforced by the BEL) need to be enforced. As a particular case, here, the set of users U may be ALL when - while processing a grant operation - the BEL determines that its protection is sufficient and therefore requests the SEL not to enforce any restriction and to possibly remove an over-encryption previously imposed.

For the sake of simplicity, in the algorithm we make use of a function *Anc_users* that, given the key $s.k$ of a vertex, returns the set of users that can derive it (i.e., can reach the vertex via a path of tokens). *Anc_users*$(s.k)$ is therefore a short-hand defined as *Anc_users*$(s.k)=\{u \in \mathcal{U} \mid s.k \in \phi^*_{\mathsf{s}}(u)\}$, which can be dynamically computed or explicitly maintained for each vertex.

Let us then see how the algorithm works. Algorithm **over-encrypt** takes a set of resources R and a set of users U as input. First, it checks whether there exists a vertex s whose key $s.k$ is used to encrypt resources in R and the set of users that can derive $s.k$ is equal to U (step 2). If such a vertex exists, intuitively, this means that resources in R are over-encrypted with a key that reflects the current *acl* of resources in R. Note that since all resources in R share the same key, it is sufficient to check the above condition on any of the resources r' in R. If this condition is evaluated to true, the algorithm terminates. Otherwise, if the resources in R are currently over-encrypted (step 2.1), they are first decrypted.[4] Then, if the set of users that should be allowed access to the resources in R by the SEL is not ALL (step 2.2), over-encryption is necessary. (No operation is executed otherwise, as U=ALL is the particular case of Delta_SEL approach discussed above.) The algorithm proceeds by checking the existence of a vertex s such that the set of users that can reach key $s.k$ (i.e., belonging to *Anc_users*$(s.k)$) corresponds to U. If such a vertex exists, then the *key* that will be used for over-encryption is set to $s.k$. Otherwise, a new key *key* is generated and a corresponding new vertex s_{new} is created, setting then $s_{new}.k$=*key*. Vertex s_{new} is then inserted in the SEL graph and tokens defined accordingly so to make $s_{new}.k$ derivable by all users in U (in terms of the graph s_{new} must be reachable by all the keys associated with users in U). To this purpose, the algorithm checks existing vertices in decreasing order of cardinality of *Anc_users*. For each s_i considered, if the set of users that can reach key $s_i.k$ (i.e., *Anc_users*$(s_i.k)$) is contained in U, a new token from $s_i.k$ to $s_{new}.k$ is generated and added. Set U is then updated by removing the set of users *Anc_users*$(s_i.k)$ that can now reach key $s_{new}.k$ thanks to the new token. This token generation process continues until U is empty, i.e., until all users have been connected to the key. Finally, all resources in R are encrypted with the designated *key* and the key assignment ϕ_{s} updated accordingly.

[4]If $\phi_{\mathsf{s}}(r)$ is no more used to protect other resources in the system, the corresponding vertex can be removed. In this case, all parents of vertex $\phi_{\mathsf{s}}(r)$ are directly connected to its children and the token catalog is changed accordingly [4]. For simplicity, we omit this operation.

4.2 Grant and revoke

Consider first a grant request **grant**(r,u) to grant user u access to resource r. The BEL level starts and regulates the update process as follows. First, $acl(r)$ is updated to include u, that is, $\mathcal{A}[u, r] = 1$ (step 1). Then, the algorithm retrieves the vertex b_i whose access key $b_i.k_{\mathsf{a}}$ is the key with which r is encrypted (step 2). If the resource's access key cannot be derived by u, then a new token from user's key $\phi_{\mathsf{b}}(u)$ to $b_i.k_{\mathsf{a}}$ is generated and added (step 3). The new token is needed to make $b_i.k_{\mathsf{a}}$ derivable by u (i.e., belonging to $\phi^*_{\mathsf{b}}(u)$). Note that the separation between derivation and access keys for each vertex allows us to add a token only giving u access to the key used to encrypt resource r, thus limiting the knowledge of each user to the information strictly needed to correctly enforce the authorization policy. Indeed, knowledge of $b_i.k_{\mathsf{a}}$ is a necessary condition to make r accessible by u. However, there may be other resources r' that are encrypted with the same key $b_i.k_{\mathsf{a}}$ and which should not be made accessible to u. Since releasing $b_i.k_{\mathsf{a}}$ would make them accessible to u, they need to be over-encrypted so to make them accessible to users in $acl(r')$ only. Then, the algorithm determines if such a set of resources R' exists (step 4). If R' is not empty, the algorithm partitions R' in sets such that each set $S \subseteq R'$ includes all resources characterized by the same *acl*, denoted acl_S (step 5.1). For each set S, the algorithm calls **over-encrypt**(S, acl_S) to demand SEL to execute an over-encryption of S for users in acl_S (step 5.2). In addition, the algorithm requests the SEL level to synchronize itself with the policy change. Here, the algorithm behaves differently depending on the encryption model assumed. In the case of Delta_SEL (step 6.1), the algorithm first controls whether the set of users that can reach the resource's access key (i.e., belonging to *Anc_users*$(b_i.k_{\mathsf{a}})$) corresponds to $acl(r)$. If so, the BEL encryption suffices and no protection is needed at the SEL level, and therefore a call **over-encrypt**$(\{r\}, \text{ALL})$ is requested. Otherwise, a call **over-encrypt**$(\{r\},acl(r))$ requests the SEL to make r accessible only to users in $acl(r)$. In the case of Full_SEL (step 6.2), this is done by calling **over-encrypt**$(r,acl(r))$, requesting the SEL to synchronize its policy so to make r accessible only by the users in $acl(r)$.

Consider now a revoke request **revoke**(r,u) to revoke from user u access to resource r. The algorithm updates $acl(r)$ to remove user u, that is, $\mathcal{A}[u, r] = 0$ (step 1). Then, it calls **over-encrypt**$(\{r\},acl(r))$ to demand SEL to make r accessible only to users in $acl(r)$ (step 2).

In terms of performance, the algorithm only requires a direct navigation of the BEL and SEL structure and it produces the identification of the requests to be sent to the server in a time which in typical scenarios will be less than the time required to send the messages to the server.

BEL

GRANT(r,u)
1. $acl(r) := acl(r) \cup \{u\}$
2. find the vertex b_i with $b_i.k_{\mathsf{a}} = \phi_{\mathsf{b}}(r)$
3. **if** $\phi_{\mathsf{b}}(r) \notin \phi^*_{\mathsf{b}}(u)$ **then**
   find the vertex b_j with $b_j.k = \phi_{\mathsf{b}}(u)$
   add_token($b_j.k$,$b_i.k_{\mathsf{a}}$)
4. find the set R' of resources r' such that
   $r' \neq r$, $\phi_{\mathsf{b}}(r') = \phi_{\mathsf{b}}(r)$, $Anc\_users(b_i.k_{\mathsf{a}}) \neq acl(r')$
5. **if** $R' \neq \emptyset$ **then**
   5.1 Partition R' in sets such that each set S contains resources with the same acl acl_S
   5.2 **for each** set S **do over-encrypt**(S,acl_S)
6. **case** encryption model **of**
   6.1 Delta_SEL:
   if $Anc\_users(b_i.k_{\mathsf{a}}) = acl(r)$ **then**
   over-encrypt($\{r\}$,ALL)
   else
   over-encrypt($\{r\}$,$acl(r)$)
   6.2 Full_SEL:
   over-encrypt($\{r\}$,$acl(r)$)

REVOKE(r,u)
1. $acl(r) := acl(r) - \{u\}$
2. **over-encrypt**($\{r\}$,$acl(r)$)

SEL

OVER-ENCRYPT(R,U)
1. let r' be a resource in R
2. **if** ($\exists$ a vertex $s : s.k = \phi_{\mathsf{s}}(r') \wedge Anc\_users(s.k) = U$) **then exit**
   else
   2.1 **if** $\phi_{\mathsf{s}}(r') \neq$ NULL **then** decrypt all $r \in R$
   2.2 **if** $U \neq$ ALL **then**
   if $\exists$ a vertex s such that $Anc\_users(s.k) = U$ **then**
   $key := s.k$
   else
   generate and add to $\mathcal{K}_{\mathsf{s}}$ a new key key
   create a new vertex s_{new}
   $s_{new}.k := key$
   while $U \neq \emptyset$
   choose a vertex s_i by considering vertices in decreasing order of cardinality of $Anc\_users(s_i.k)$
   if $Anc\_users(s_i.k) \subseteq U$ **then**
   add_token($s_i.k$,$s_{new}.k$)
   $U := U - Anc\_users(s_i.k)$
   for each $r \in R$ **do**
   $\phi_{\mathsf{s}}(r) := key$
   encrypt r with key

Figure 5: Algorithms for granting and revoking authorizations

4.3 Example

We present an example that illustrates the behavior of the revoke and grant operations described above. We assume the initial configuration depicted in Figure 4. Figure 6 illustrates the evolution of the encryption policies at both BEL and SEL in the Full_SEL and Delta_SEL modes, upon the following sequence of policy changes. Here, dashed edges at the BEL level correspond to the tokens added by the **grant** algorithm.

- **grant**(r_5,D): key $b_6.k_{\mathsf{a}}$ used to encrypt r_5 does not belong to $\phi^*_{\mathsf{b}}(D)$. The data owner therefore adds a BEL token $t_{4,6}$ in the BEL. Since $b_6.k_{\mathsf{a}}$ is also used to encrypt resources r_6 and r_7, these resources have to be over-encrypted in such a way that they are accessible only to users A, B, and C. In the Delta_SEL scenario, **over-encrypt** creates a new vertex s_6 for resources r_6 and r_7. The protection of resource r_5 at BEL level is instead sufficient and no over-encryption is needed. In the Full_SEL scenario resources r_6 and r_7 are already correctly protected and r_5 is instead over-encrypted with key $s_9.k$.

- **revoke**(r_2,C): user C is removed from $acl(r_2)$. Since now this acl becomes empty, resource r_2 has to be over-encrypted with a key that no user can compute. Consequently, both in the Delta_SEL and in the Full_SEL scenario, a new vertex $s_\top$ is created and its key is used to protect r_2.

- **grant**(r_4,E): $b_7.k_{\mathsf{a}}$ used to encrypt r_4 does not belong to $\phi^*_{\mathsf{b}}(E)$. The data owner therefore adds a BEL token $t_{5,7}$ allowing E to compute $b_7.k_{\mathsf{a}}$. Since $b_7.k_{\mathsf{a}}$ is also used to protect r_3, we need to call **over-encrypt**(r_3,$\{C, D\}$). In the Delta_SEL scenario, **over-encrypt** creates a new vertex s_7 for r_3, while the BEL protection of resource r_4 is sufficient and no over-encryption is needed. In the Full_SEL scenario, r_3 is already correctly protected and r_4 is over-encrypted with $s_{10}.k$, which only C, D, and E can derive.

- **grant**(r_6,D): $b_6.k_{\mathsf{a}}$ used to encrypt r_6 already belongs to $\phi^*_{\mathsf{b}}(D)$, and therefore the BEL encryption policy does not change. However, since r_6 shares its BEL access key with r_5 and r_7, the SEL encryption policy may need to be updated. In particular, both the Delta_SEL and the Full_SEL scenario already correctly enforce the policy w.r.t. r_5 and r_7. In the Delta_SEL scenario, the over-encryption of r_6 is removed because the BEL encryption policy is sufficient, while in the Full_SEL scenario r_6 is over-encrypted with a new key $s_9.k$.

4.4 Correctness

We now prove that the algorithm implementing the grant and revoke operations preserves the correctness of the encryption policy.

THEOREM 4.1 (CORRECTNESS). *Let $\mathcal{E}_{\mathsf{b}} = (\tau^*_{\mathsf{b}}, \phi_{\mathsf{b}})$ be an encryption policy at the BEL level, $\mathcal{E}_{\mathsf{s}} = (\tau^*_{\mathsf{s}}, \phi_{\mathsf{s}})$ be an encryption policy at the SEL level, and $\mathcal{A}$ an access control policy such that $\langle \mathcal{E}_{\mathsf{b}}, \mathcal{E}_{\mathsf{s}} \rangle \Longleftrightarrow \mathcal{A}$. Algorithms in Figure 5 generate a new $\mathcal{E}_{\mathsf{b}}{}' = (\tau'^*_{\mathsf{b}}, \phi'_{\mathsf{b}})$, $\mathcal{E}_{\mathsf{s}}{}' = (\tau'^*_{\mathsf{s}}, \phi'_{\mathsf{s}})$, and $\mathcal{A}'$ such that $\langle \mathcal{E}_{\mathsf{b}}{}', \mathcal{E}_{\mathsf{s}}{}' \rangle \Longleftrightarrow \mathcal{A}'$.*

PROOF. (sketch)
Since we start with a BEL encryption policy and a SEL encryption policy that satisfy Definition 5 (as discussed in Section 3.1.3), we will therefore consider only users and resources for which the encryption policies change. Grant and revoke are based on the correct enforcement of the over-encryption operation. We then examine it first.

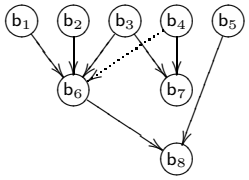
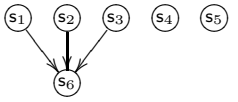
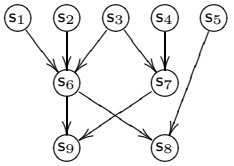
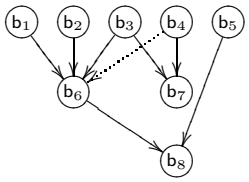
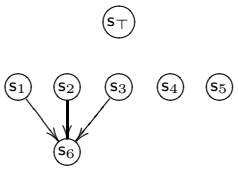
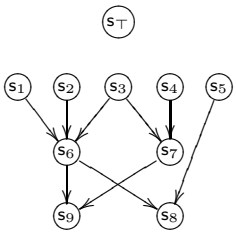
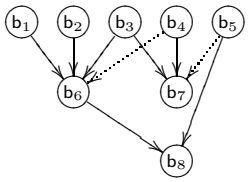
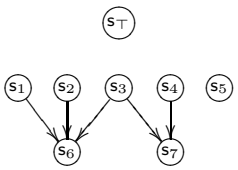
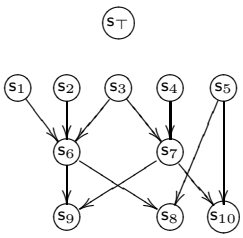
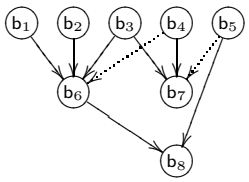
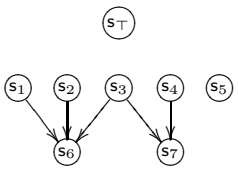
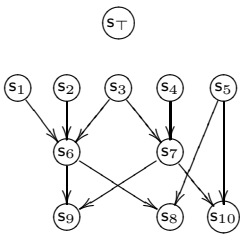
BEL	Delta_SEL	Full_SEL
grant(r_5,D)	**over-encrypt**(r_6,r_7,ABC) **over-encrypt**(r_5,ALL)	**over-encrypt**(r_6,r_7,ABC) **over-encrypt**(r_5,$ABCD$)
		
r | $\phi_{\mathsf{s}}(r)$ r_7,r_6 | $\mathsf{s}_6.k$ r_1,...,r_5 | NULL r_8 | NULL		r | $\phi_{\mathsf{s}}(r)$ r_1,r_2 | $\mathsf{s}_3.k$ r_3,r_4 | $\mathsf{s}_7.k$ r_5 | $\mathsf{s}_9.k$ r_6,r_7 | $\mathsf{s}_6.k$ r_8 | $\mathsf{s}_8.k$
revoke(r_2,C)	**over-encrypt**(r_2,$\emptyset$)	**over-encrypt**(r_2,$\emptyset$)
		
r | $\phi_{\mathsf{s}}(r)$ r_2 | $\mathsf{s}_\top.k$ r_6,r_7 | $\mathsf{s}_6.k$ r_1 | NULL r_3,r_4,r_5 | NULL r_8 | NULL		r | $\phi_{\mathsf{s}}(r)$ r_1 | $\mathsf{s}_3.k$ r_2 | $\mathsf{s}_\top.k$ r_3,r_4 | $\mathsf{s}_7.k$ r_5 | $\mathsf{s}_9.k$ r_6,r_7 | $\mathsf{s}_6.k$ r_8 | $\mathsf{s}_8.k$
grant(r_4,E)	**over-encrypt**(r_3,CD) **over-encrypt**(r_4,ALL)	**over-encrypt**(r_3,CD) **over-encrypt**(r_4,CDE)
		
r | $\phi_{\mathsf{s}}(r)$ r_2 | $\mathsf{s}_\top.k$ r_3 | $\mathsf{s}_7.k$ r_6,r_7 | $\mathsf{s}_6.k$ r_1 | NULL r_4,r_5 | NULL r_8 | NULL		r | $\phi_{\mathsf{s}}(r)$ r_1 | $\mathsf{s}_3.k$ r_2 | $\mathsf{s}_\top.k$ r_3 | $\mathsf{s}_7.k$ r_4 | $\mathsf{s}_{10}.k$ r_5 | $\mathsf{s}_9.k$ r_6,r_7 | $\mathsf{s}_6.k$ r_8 | $\mathsf{s}_8.k$
grant(r_6,D)	**over-encrypt**(r_7,ABC) **over-encrypt**(r_6,ALL)	**over-encrypt**(r_7,$ABCD$) **over-encrypt**(r_6,$ABCD$)
		
r | $\phi_{\mathsf{s}}(r)$ r_2 | $\mathsf{s}_\top.k$ r_3 | $\mathsf{s}_7.k$ r_7 | $\mathsf{s}_6.k$ r_1 | NULL r_4,r_5,r_6 | NULL r_8 | NULL		r | $\phi_{\mathsf{s}}(r)$ r_1 | $\mathsf{s}_3.k$ r_2 | $\mathsf{s}_\top.k$ r_3 | $\mathsf{s}_7.k$ r_4 | $\mathsf{s}_{10}.k$ r_5,r_6 | $\mathsf{s}_9.k$ r_7 | $\mathsf{s}_6.k$ r_8 | $\mathsf{s}_8.k$

Figure 6: An example of grant and revoke operations

Over-encrypt. We need to prove that **over-encrypt**(R, U) possibly encrypts all resources in R with a key in such a way that a user u' can derive such a key if and only if $u' \in U$. The only case we need to consider is when the set of users U is different from ALL (when $U =$ ALL, resources in R are not needed to be over-encrypted). Then, if the condition in the first **if** statement (step 2) is evaluated to true, resources in R are already correctly protected and since the algorithm terminates, the result is correct. Otherwise, resources in R are first possibly decrypted and then encrypted with the correct key $s.k$ or with the new generated key, which is assigned to a new vertex s_{new}. In the first case, the result is trivially correct. In the second case, correctness is guaranteed by the construction of the tokens in the **while** statement (a user will reach $s_{new}.k$ iff she belongs to U).

Grant. $\langle \mathcal{E}_\mathsf{b}', \mathcal{E}_\mathsf{s}' \rangle \Longrightarrow \mathcal{A}'$ (soundness)
Consider user u and resource r. From step 3, it is easy to see that $\phi'_\mathsf{b}(r) = \phi_\mathsf{b}(r) \in \phi'^*_\mathsf{b}(u)$. From step 6 and by the correctness of **over-encrypt**, either $\phi'_\mathsf{s}(r) =$ NULL or r is over-encrypted with a key such that $\phi'_\mathsf{s}(r) \in \phi'^*_\mathsf{s}(u)$ (user u is included in the current $acl(r)$). Since $\phi'_\mathsf{b}(r) \in \phi'^*_\mathsf{b}(u)$ and $\phi'_\mathsf{s}(r) \in \phi'^*_\mathsf{s}(u)$, we have that $\mathcal{A}'[u, r] = 1$.

Consider now the set of resources R' and suppose that R' is not empty. For each subset S of R', user u can now derive the key used to encrypt such a set of resources. This implies that $\forall r' \in S$, $\phi'_\mathsf{b}(r') = \phi_\mathsf{b}(r') \in \phi'^*_\mathsf{b}(u)$. However, by the correctness of **over-encrypt**, a call **over-encrypt**(S, acl_S) guarantees that all resources r' in S are over-encrypted with a key such that $\forall r' \in S, \phi'_\mathsf{s}(r') \notin \phi'^*_\mathsf{s}(u)$ because acl_S does not include user u.

$\langle \mathcal{E}_\mathsf{b}', \mathcal{E}_\mathsf{s}' \rangle \Longleftarrow \mathcal{A}'$ (completeness)
Consider user u and resource r. From step 1, we have that $\mathcal{A}'[u, r] = 1$. From step 3, it is easy to see that $\phi'_\mathsf{b}(r) = \phi_\mathsf{b}(r) \in \phi'^*_\mathsf{b}(u)$. Also, from step 6 and by the correctness of **over-encrypt**, either $\phi'_\mathsf{s}(r) =$ NULL or r is over-encrypted with a key such that $\phi'_\mathsf{s}(r) \in \phi'^*_\mathsf{s}(u)$.

Revoke. $\langle \mathcal{E}_\mathsf{b}', \mathcal{E}_\mathsf{s}' \rangle \Longrightarrow \mathcal{A}'$ (soundness)
Consider user u and resource r. A call **over-encrypt**$(\{r\}, acl(r))$ is requested to demand the SEL to make r accessible only to users in the current $acl(r)$. We know that $\phi'_\mathsf{b}(r) \in \phi'^*_\mathsf{b}(u)$. Also, from the **over-encrypt** correctness, it is easy to see that $\phi'_\mathsf{s}(r) \notin \phi'^*_\mathsf{s}(u)$.

$\langle \mathcal{E}_\mathsf{b}', \mathcal{E}_\mathsf{s}' \rangle \Longleftarrow \mathcal{A}'$ (completeness)
Consider user u and resource r. From step 1 we have that $\mathcal{A}[u, r] = 0$. The subsequent call **over-encrypt**$(\{r\}, acl(r))$ makes resource r no more accessible to user u because r is over-encrypted with a key that is no more derivable by u (this property is a consequence of the **over-encrypt** correctness), that is, $\phi'_\mathsf{b}(r) \in \phi'^*_\mathsf{b}(u)$ and $\phi'_\mathsf{s}(r) \notin \phi'^*_\mathsf{s}(u)$. □

5. PROTECTION EVALUATION

Since the BEL and SEL are corrected at initialization time, the correctness of the algorithm ensures that the encryption policy $\mathcal{E}$ correctly models the authorization policy $\mathcal{A}$. In other words, at any point in time, users will be able to access only resources for which they have - directly or indirectly - the necessary keys both at the BEL and at the SEL level.

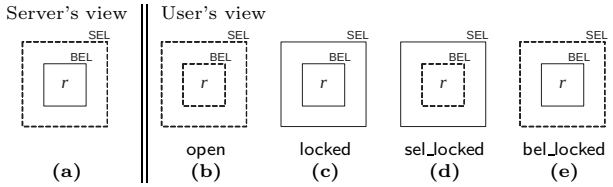


Figure 7: Possible views on resource r

The key derivation function adopted is proved to be secure [4]. We also assume that all the encryption functions and the tokens are robust and cannot be broken, even combining the information available to many users. Moreover, we assume that each user correctly manages her keys, without the possibility for a user to steal keys from another user.

It still remains to evaluate whether the approach is vulnerable to attacks from users who access and store all information offered by the server, or from *collusion* attacks, where different users (or a user and the server) combine their knowledge to access resources they would not otherwise be able to access. Note that for collusion to exist, both parties should gain in the exchange (as otherwise they will not have any incentive in colluding). We now discuss possible information exposure, with the conservative assumption that users are not oblivious (i.e., they have the ability to store and keep indefinitely all information they were entitled to access). In the discussion we first refer to the Full_SEL approach; we will then analyze the Delta_SEL approach.

To be able to model exposure, we first start by examining the different views that one can have on a resource r. To do so, we exploit a graphical notation with resource r in the center and with fences around r denoting the barriers to the access imposed by the knowledge of the keys used for r's encryption at the BEL (inner fence, $\phi_\mathsf{b}(r)$) and at the SEL (outer fence, $\phi_\mathsf{s}(r)$). The fence is continuous if there is no knowledge of the corresponding key (the barrier cannot be passed) and it is discontinuous otherwise (the barrier can be passed).

Figure 7 illustrates the different views that can exist on the resource. On the left, Figure 7(a), there is the view of the SEL server itself, which knows the key at the SEL level but does not have access to the key at the BEL level. On the right, there are the different possible views of users, for whom the resource can be:

- open: the user knows the key at the BEL level as well as the key at the SEL level (Figure 7(b));
- locked: the user knows neither the key at the BEL level nor the key at the SEL level (Figure 7(c));
- sel_locked: the user knows only the key at the BEL level but does not know the key at the SEL level (Figure 7(d));
- bel_locked: the user knows only the key at the SEL level but does not know the one at the BEL level (Figure 7(e)). Note that this latter view corresponds to the view of the server itself.

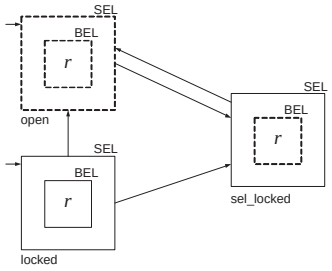


Figure 8: View transitions in the Full_SEL

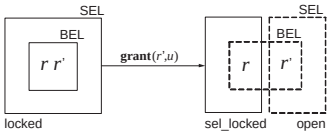


Figure 9: From locked to sel_locked views

Figure 10: Classification of users for a resource r

By the correctness of the approach (Theorem 4.1), the open view corresponds to the view of authorized users, while the remaining views correspond to the views of non authorized users.

Before analyzing exposure, we illustrate how the different views on a resource can be produced by authorization changes (and consequent over-encryption).

5.1 View evolution

In the Full_SEL approach, at initialization time, BEL and SEL are completely synchronized. Then, for each user, a resource is protected by both keys or by neither: authorized users will have the open view, while non authorized users will have the locked view. Figure 8 summarizes the possible view transitions starting from these two views.

Let us first examine the evolution of the open view. Since resources at the BEL level are not re-encrypted, the view of an authorized user can change only if the user is revoked the authorization. In this case, the resource is over-encrypted at the SEL level, then becoming sel_locked for the user. The view could be brought back to be open if the user is granted the authorization again (i.e., over-encryption is removed).

Let us now examine the evolution of the locked view. First, we note that - for how the SEL is constructed and maintained in the Full_SEL approach - it cannot happen that the SEL grants a user an access that is blocked at the BEL level, and therefore the bel_locked view can never be reached. The view can instead change to open, in case the user is granted the authorization to access the resource; or to sel_locked, in case the user is given the access key at the BEL level but she is not given that at the SEL level. This latter situation can happen if the release of the key at the BEL level is necessary to make accessible to the user another resource r' that is, at the BEL level, encrypted with the same key as r. To illustrate, suppose that at initialization time resources r and r' are both encrypted with the same key and they are not accessible by user u (whose view on the resources is locked). The leftmost view in Figure 9 illustrates this situation. Suppose then that u is granted the authorization for r'. To make r' accessible at the BEL level, a token is added to make $\phi_{\mathsf{b}}(r)$ derivable by u, where however $\phi_{\mathsf{b}}(r)$=$\phi_{\mathsf{b}}(r')$. Hence, r' will be over-encrypted at the SEL level and $\phi_{\mathsf{s}}(r')$ made derivable by u. The resulting situation is illustrated in Figure 9, where r' is open and r results sel_locked.

5.2 Exposure risk

Collusion can take place every time two entities, combining their knowledge (i.e., the keys known to them) can acquire knowledge that *neither* of them has access to. Therefore, users having the open view need not be considered as they have nothing to gain in colluding (they already access r). By the same line of reasoning, also users having the locked view need not be considered as they have nothing to offer (and therefore no one will have to gain in colluding with them). Also, recall that in the Full_SEL approach, for what said in the previous subsection, nobody (but the server) can have a bel_locked view. This leaves us only with users having the sel_locked view. Since users having the same views will not gain anything in colluding, the only possible collusion can happen between the server (who has a bel_locked view) and a user who has a sel_locked view. In this situation, the knowledge of the server allows lowering the outer fence, while the knowledge of the user allows lowering the inner fence: merging their knowledge, they would then be able to bring down both fences and enjoy the open view on the resource.

We know, from the possible transitions between views, that there are two reasons for which a user can have the sel_locked view on a resource.

- *Past_acl*: the user was previously authorized to access the resource and the authorization was then revoked from her (transition from open to sel_locked in Figure 8). By colluding with the server, in this case the user would get access to a resource she previously had authorization for. Since we assume users to be non oblivious, the user - while not currently having ability to access the resource - can have it cached at her side already. Then she has no gain in colluding with the server. It is therefore legitimate to consider this case ineffective with respect to collusion risks. [5]

- *Policy_split*: the user has been granted the authorization for another resource that was, at the initialization time, encrypted with the same key as r, leaving r sel_locked. In this situation (from locked to sel_locked), the user has never had access to r and should still not have it; therefore there is indeed exposure to collusion.

In other words, the risk of collusion arises on resources for which a user holds a sel_locked view and the user never had the authorization to access the resource (i.e., the user never belonged to the *acl* of the resource). This observation provides a way to measure the collusion risk to which a resource is exposed. Figure 10 provides a graphical illustration of the set of users and their possible relationship with the

[5]We assume, without loss of generality, that any time a resource is updated, the data owner encrypts it with the right BEL key.

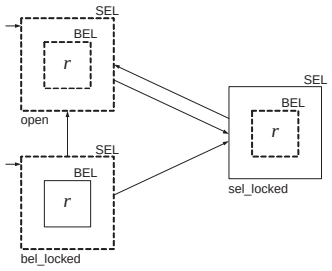


Figure 11: View transitions in the Delta_SEL

resource. The outermost set represents all the users. For a subset of them, denoted as *Bel_accessible*, the resource might be not protected at the BEL level (there is no fence as they know $\phi_b(r)$). Among them, there are the users, denoted as *Past_acl*, who had in the past the authorization to access r, some of whom (those in $acl(r)$) may still hold the authorization. The users representing a collusion risk for the resource are all those that belong to *Bel_accessible* and do not belong to *Past_acl*.

Besides collusion between different parties, we also need to consider the risk of exposure due to a single user (or server) merging her own views on a resource at different points in time. It is easy to see that, in the Full_SEL approach, where all non authorized users start with a locked view on the resource (and transitions are as illustrated in Figure 8), there is no risk of exposure. Trivially, if the user is released the key at the SEL level (i.e., it is possible for her to bring down the lower fence) it is because the user has the authorization for r at some point in time and therefore she is (or has been) authorized for the resource. There is therefore no exposure risk.

As for the Delta_SEL approach, we start by noting that users not authorized to see a resource have, at initial time, the bel_locked view on it. From there, the view can evolve to be sel_locked or open (in case the user is given the authorization for the resource). View transitions are illustrated in Figure 11. It is easy to see that, in this case, a single user by herself can then hold the two different views: sel_locked and bel_locked. In other words a (planning-ahead) user could retrieve the resource at initial time, when she is not authorized, getting and storing at her side r's bel_locked view. If, at a later point in time the user is released $\phi(r)$ to make accessible to her another resource r', she will acquire the sel_locked view on r. Merging this with the past bel_locked view, she can enjoy the open view on r. Note that the set of resources potentially exposed to a user coincides with the resources exposed to collusion between that user and the server in the Full_SEL approach.

It is important to note that in both cases (Full_SEL and Delta_SEL), exposure is limited to resources that have been involved in a policy split to make other resources, encrypted with the same BEL key, available to the user. Exposure is therefore limited and well identifiable. This allows the owner to possibly counteract it via explicit selective re-encryption or by proper design (as discussed in the next section).

The collusion analysis clarifies why we did not consider the third possible encryption scenario illustrated in Section 3. In this scenario, all users non authorized to access a resource would always have the sel_locked view on it and could potentially collude with the server. The fact that the BEL key is the same for all resources would make all the resources exposed (as the server would need just one key to be able to access them all).

5.3 Design considerations

From the analysis above, we can make the following observations on the Delta_SEL and the Full_SEL approaches.

- *Exposure protection*. The Full_SEL approach provides superior protection, as it reduces the risk of exposure, which is limited to collusion with the server. By contrast, the Delta_SEL approach exposes also to single (planning-ahead) users.
- *Performance*. The Delta_SEL approach provides superior performance, as it imposes over-encryption only when required by a change in authorizations. By contrast, the Full_SEL approach always imposes a double encryption on the resources, and therefore an additional load.

From these observations we can draw some criteria that could be followed by a data owner when choosing between the use of Delta_SEL or Full_SEL. If the data owner knows that:

- the access policy will be relatively static, or
- sets of resources sharing the same *acl* at initialization time represent a strong semantic relationship rarely split by policy evolution, or
- resources are grouped in the BEL in fine granularity components where most of the BEL nodes are associated with a single or few resources,

then the risk of exposing the data to collusion is limited also in the Delta_SEL approach, which can then be preferred for performance reasons.

By contrast, if authorizations have a more dynamic and chaotic behavior, the Full_SEL approach can be preferred to limit exposure due to collusion (necessarily involving the server). Also, the collusion risk can be minimized by a proper organization of the resources to reduce the possibility of policy splits. This could be done either by producing a finer granularity of encryption and/or better identifying resource groups characterized by a persistent semantic affinity (in both cases, using in the BEL different keys for resources with identical *acl*).

6. RELATED WORK

Previous work close to our is in the area of "database-as-a-service" paradigm [10, 11], which considers the problem of database outsourcing. Its intended purpose is to enable data owners to outsource distribution of data on the Internet to service providers. The majority of existing efforts on this topic focuses on techniques allowing the execution of queries on encrypted outsourced data, trying to support all SQL clauses and different kinds of conditions over attributes [2, 7, 10, 11]. In general, these approaches are based on indexing information stored together with the encrypted data. Such indexes are used to select the data to be returned in response to a query, without need of decrypting the data. One of the major challenges in the development of indexing techniques is the trade off between query efficiency and

exposure to inference and linking attacks that strongly depends on the attacker's prior knowledge [7]. A related effort in [14] focuses on the design of mechanisms for protecting the integrity and authenticity of data from both malicious outsider attacks and the service provider itself.
Other proposals have investigated the use of partitioning, adopting a distributed architecture to allow an organization to outsource its data management to two untrusted servers while preserving data privacy [1].
In addition to the application-based approaches abovementioned, hardware-based approaches to the problem of secure outsourced storage have also been investigated [6]. Here, the basic idea is to use a special security hardware component, which can support secure computations at both client and server sides.

A few research efforts have directly tackled the issues of access control in an outsourced scenario. In [13] the authors present a framework for enforcing access control on published XML documents by using different cryptographic keys over different portions of the XML tree and by introducing special metadata nodes in the structure. Our work is complementary to this proposal, as it looks at the different problem of enforcing policy changes.

On a different line of related work, other approaches have investigated the hierarchical-based access control in the context of distributed environments and pay-tv [5, 17]. While some commonalities can be identified (e.g., the use of cryptographic key-based hierarchical schemes), the outsourced data scenario presents peculiar characteristics that require the development of new solutions.

7. CONCLUSIONS

There is an emerging trend towards scenarios where resource management is outsourced to an external service providing storage capabilities and high-bandwidth distribution channels. In this context, selective release requires enforcing measures to protect the resource confidentiality from both unauthorized users as well as "honest-but-curious" servers. Current solutions provide protection by exploiting encryption in conjunction with proper indexing capabilities, but suffer from limitations requiring the involvement of the owner every time selective access is to be enforced or the access policy is modified. In this paper we have put forward the idea of enforcing the authorization policy by using a two-layer selective encryption. Our solution offers significant benefits in terms of quicker and less costly realization of authorization policy updates and general efficiency of the system (replication of resources can carry along the policy itself). We believe these benefits to be crucial for the success of emerging scenarios characterized by a huge number of resources of considerable size and that have to be distributed in a selective way to a variety of users.

8. ACKNOWLEDGMENTS

This work was supported in part by the European Union under contract IST-2002-507591, and by the Italian Ministry of Research, within programs FIRB, under project "RBNE05FKZ2", and PRIN 2006, under project "Basi di dati crittografate" (2006099978). The work of Sushil Jajodia was partially supported by the National Science Foundation under grants CT-0627493, IIS-0242237, and IIS-0430402.

9. REFERENCES

[1] G. Aggarwal et al. Two can keep a secret: a distributed architecture for secure database services. In *Proc. of CIDR 2005*, Asilomar, CA, Jan. 2005.

[2] R. Agrawal, J. Kierman, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proc. of ACM SIGMOD 2004*, Paris, France, June 2004.

[3] S. Akl and P. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM TOCS*, 1(3):239–248, August 1983.

[4] M. Atallah, K. Frikken, and M. Blanton. Dynamic and efficient key management for access hierarchies. In *Proc. of the 12th ACM CCS05*, Alexandria, VA, USA, Nov. 2005.

[5] J. Birget, X. Zou, G. Noubir, and B. Ramamurthy. Hierarchy-based access control in distributed environments. In *Proc. of IEEE International Conference on Communications*, Finland, June 2002.

[6] L. Bouganim and P. Pucheral. Chip-secured data access: confidential data on untrusted servers. In *Proc. of the 28th VLDB Conference*, Hong Kong, China, August 2002.

[7] A. Ceselli, E. Damiani, S. D. C. di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM TISSEC*, 8(1):119–152, Feb. 2005.

[8] J. Crampton, K. Martin, and P. Wild. On key assignment for hierarchical access control. In *Proc. of the 19th IEEE CSFW'06*, Venice, Italy, July 2006.

[9] E. Damiani et al. An experimental evaluation of multi-key strategies for data outsourcing. In *Proc. of the 22nd IFIP TC-11 International Information Security Conference*, South Africa, May 2007.

[10] H. Hacigümüs, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proc. of 18th ICDE*, San Jose, CA, USA, Feb. 2002.

[11] H. Hacigümüs, B. Iyer, S. Mehrotra, and C. Li. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of the ACM SIGMOD 2002*, Madison, Wisconsin, USA, June 2002.

[12] S. Jajodia, P. Samarati, M. Sapino, and V. Subrahmanian. Flexible support for multiple access control policies. *ACM TODS*, 26(2):214–260, June 2001.

[13] G. Miklau and D. Suciu. Controlling access to published data using cryptography. In *Proc. of the 29th VLDB conference*, Berlin, Germany, Sept. 2003.

[14] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced database. In *Proc. of the 11th NDSS04*, San Diego, CA, USA, Feb. 2004.

[15] R. Sandhu. On some cryptographic solutions for access control in a tree hierarchy. In *Proc. of the FJCC'87*, Dallas, TX, USA, Oct. 1987.

[16] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. On the twofish key schedule. In *Selected Areas in Cryptography*, June 1998.

[17] Y. Sun and K. Liu. Scalable hierarchical access control in secure group communications. In *Proc. of the IEEE Infocom*, Hong Kong, China, March 2004.

[18] XML Encryption Syntax and Processing, W3C Rec. http://www.w3.org/TR/xmlenc-core/, Dec. 2002.