Supporting Delegation in Outsourced ICA Process

Sabrina De Capitani di Vimercati[1][0000−0003−0793−3551], Sara Foresti[1][0000−0002−1658−6734], Stefano Paraboschi[2][0000−0003−0399−1738], Sara Petrilli[1], and Pierangela Samarati[1][0000−0001−7395−4620]

[1] Università degli Studi di Milano, Italy
{sabrina.decapitani,sara.foresti,pierangela.samarati}@unimi.it;
sara.petrilli@studenti.unimi.it
[2] Università degli Studi di Bergamo, Italy
stefano.paraboschi@unibg.it

Abstract. We consider the problem of enforcing corporate governance control relying on cloud-based services. Extending previous work, we focus in particular on the support of delegation of the director privileges, enabling their dynamic and temporary assignment to a vice-director. Like previous work, our control relies on encrypted tags, which are here extended addressing the challenges introduced by dynamic delegation which operates on a time dimension orthogonal to the corporate governance control process. Our solution enables delegation while ensuring a vice-director to enjoy the director privileges only when delegation is active and not to operate as director for operations the vice-director has processed as employee (separation of duties). Our tag construction ensures integrity of the dynamic delegation control and protection against tag tampering.

Keywords: Cloud-based services, outsourcing, internal controls and audit process, delegation, separation of duties

1 Introduction

Cloud-based applications and services represent today a convenient alternative to on-premises solutions for the management of applications and processes, due to their scalability, efficiency, and cost benefits. Adoption of cloud-based solutions for sensitive or critical applications requires, however, particular care, to ensure confidentiality and integrity of the data and process are properly considered. In this paper, we consider the enforcement of corporate governance control with cloud-based services and, specifically of the *Internal Controls and Audit* (ICA) functions, aimed at verifying the compliance of the operations generated and elaborated within an organization with internal rules, regulations, and laws. More concretely, we consider a three-phase ICA process, which is the most common in companies that have to comply with market regulations, like bank and financial institutions. In this context, companies are organized in units, which

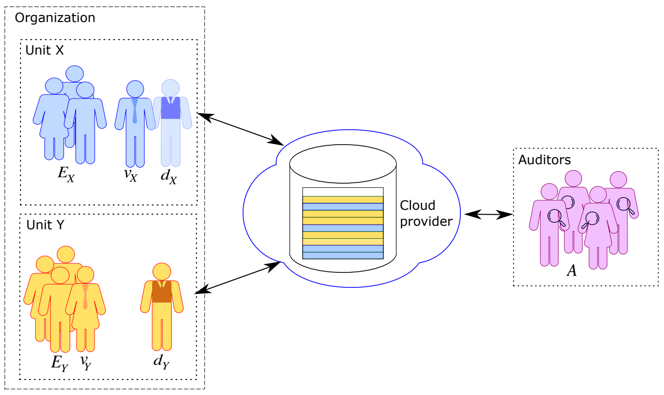


Fig. 1: Reference scenario

generate and process operations (e.g., a bank operating through branches and processing money withdrawal and deposit), and each operation goes through three phases. Each phase is under the control of a different subject, who is in charge of verifying different aspects of the operation and produces a report for the phase. The first (*employee*) phase is executed by an employee of the unit where the operation has been processed. The second (*director*) phase is executed by the director of the unit where the operation has been processed. The third (*auditor*) and final phase is executed by an independent auditor external to the company. Operations and reports should be visible only to auditors and to the employees and director of the unit where the operation has been processed, and each report can be generated only by a subject with the role (employee, director, auditor) for the report. Also, a report can be updated only by the subject who generated it, and only during the corresponding phase.

The approach in [4] for enforcing the ICA process, while relying on cloud-based services, assumes all subjects to be available for the execution of their phase. However, since each unit has one director only, absence of the director prevents execution of the second phase of the ICA process for all the operations processed at the unit, which remain therefore blocked as a phase cannot start before the completion of the previous one. While the block in the operativity of the unit caused by the absence of its director can be easily solved when managing the ICA process on-premises by simply delegating an employee of the unit (the vice-director), enforcing delegation in the cloud introduces complications since the cloud provider is assumed to simply provide services and execute requested actions while remaining unaware of the ICA process.

In this paper, we build on the approach in [4], extending and revising it to enable delegation of the director privileges. With reference to Figure 1, the goal is to support dynamic delegation enabling a vice-director (e.g., v_X in the figure) to take on the director privileges for the operations in their unit. Our solution leverages tags associated with operations and units to support the execution of the ICA phases, which corresponds to write actions executed at the server and regulated by such tags (which enable the enforcement of access regulations). Our approach supports the intrinsically dynamic nature of delegation without the need to rewrite the tags of operations and permits the vice-director to operate on behalf of the director only when delegation is active, independently from the status of the operation at the time of activation/deactivation of delegation (e.g., on operations for which the first phase terminated before delegation). Also, our approach enforces separation of duties, preventing the vice-director from performing the first and second phase of control on a same operation, thus guaranteeing the involvement of three different subjects in the ICA process of each operation (independently from who performed the first phase of control). Our tag construction ensures integrity of the dynamic delegation control and protection against tag tampering. The main advantage of our solution is the direct support of the ICA process, including delegation, while leveraging basic services of the cloud provider.

The remainder of this paper is organized as follows. Section 2 illustrates the basic concepts on which our work is based. Section 3 characterizes the aspects to take into account for supporting dynamic delegation of the director's role. Section 4 describes our solution for supporting delegation of the director's privileges in the ICA process. Section 5 illustrates the pseudocode of the procedures implementing our solution. Section 6 discusses related work. Finally, Section 7 concludes the paper. The Appendix shows that our procedures correctly enforce the write controls on reports and tags.

2 ICA Process in the Cloud

In this section, we illustrate the basic concepts of the approach proposed in [4] for enforcing the ICA process in the cloud. Since our focus is on the management of delegation, we limit the concepts to those affected by delegation and simplify notation to refer to a single unit. The approach leverages symmetric encryption and a hierarchical organization of keys associated with the different subjects (and groups thereof) of the ICA process. Each individual subject s (i.e., employee e, director d, auditor a), the set E of employees of each unit (with $d \notin E$), and the set A of auditors is associated with a key known also to the provider. Hierarchical key organization enables each individual subject s (and the provider) to derive the key of the group to which s belongs. Denoting key assignment with $\phi(s)$ and key derivation with $\rightsquigarrow$, this is formally expressed as $\forall e \in E, a \in A : \phi(e) \rightsquigarrow \phi(E), \phi(a) \rightsquigarrow \phi(A)$. In the following, we will also use k_s to denote the key assigned to a subject s and known to the provider, that is, $\phi(s) = k_s$.[3]

[3] While in the original model a prime superscript was used to denote encryption keys shared with the provider (in contrast to the keys known only to subjects and used for reading and writing reports) and their assignment function, focusing only on the controls on tags regulating write operations, which are based on keys shared with the provider, in this paper we simplify notation and omit such superscript.

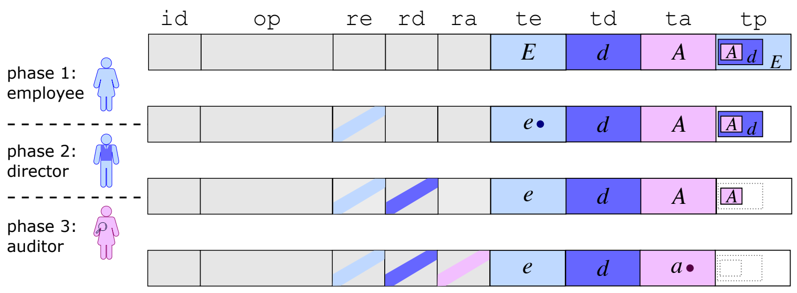


Fig. 2: Evolution of the reports and tags for an operation

Regulation of write privileges on reports relies on the use of tags, which are random values encrypted with keys shared with the service provider. For each unit, a (precomputed) strip of tags is defined. This strip is attached to every operation of the unit at generation time, and evolves as the ICA process for the operation progresses. The strip has four tags: a tag for each report (`te` for employee report `re`, `td` for director report `rd`, and `ta` for auditor report `ra`), and a phase tag (`tp`). The random values encrypted in the tags are all different (to prevent subjects from operating outside their role) and each report tag is encrypted with the key of the subjects allowed to write the corresponding report (k_E for `te`, k_d for `td`, and k_A for `ta`). The phase tag has three layers of encryption, each using the key of the subjects authorized to perform one of the three phases. More precisely, the random value within the phase tag is encrypted (as an onion) with k_A, then k_d, then k_E. The phase tag evolves as the ICA process progresses regulating the start and end of each phase. Intuitively, a subject will be authorized to operate on a report only if proving ability to decrypt both the report tag and the phase tag. Hence, when the process for an operation starts, only employees will be able to operate. When the employee phase is completed, the outer layer of the phase tag is peeled, enabling the second phase in which only the director would be able to operate. Similarly, when the second phase completes and the third is enabled, only auditors can operate.

Using dot notation to refer to the different fields of an operation (including the operation identifier `id`, the operation content `op`, reports, and tags), a subject s will be authorized for a write operation on a report $o.\texttt{r}*$, with $* \in \{\texttt{e,d,a}\}$, only if proving ability to decrypt the corresponding report tag $o.\texttt{t}*$ and the phase tag $o.\texttt{tp}$. Denoting with W the write actions to be authorized and with λ the function assigning keys to tags (i.e., identifying the keys used for encrypting tags), write control is formally captured by the following property.

Property 1 (Write control). For each subject s, operation o, and report $o.\texttt{r}*$, with $* \in \{\texttt{e,d,a}\}$: $\texttt{write}(s,o.\texttt{r}*) \in \mathrm{W}$ iff $\phi(s) \rightsquigarrow \lambda(o.\texttt{t}*) \wedge \phi(s) \rightsquigarrow \lambda(o.\texttt{tp})$.

Figure 2 summarizes the evolution of reports and tags for an operation. In the figure, for simplicity, we specify for each tag the subjects who can derive the corresponding encryption key. Note that when an employee e (auditor a, resp.) starts the employee phase (auditor phase, resp.), they overwrite the value for `te` (`ta`, resp.) with a new random value encrypted with their own key k_e (k_a, resp.). This prevents updates to the report by other employees (auditors, resp.). In the figure, a bullet denotes this change in the random value of a tag.

3 Delegation in the ICA Process

The execution of the ICA process regulated by tags as above relies on the presence of all the subjects authorized to perform its three phases. While the first (employee) phase and last (auditor) phase can be executed by any of the subjects operating in the required role (i.e., any employee or any auditor, resp.), the second phase can be executed only by the director. If the director is temporarily unavailable (e.g., for a sick leave) the ICA process would remain blocked for all operations for which the second phase has not been performed. Our goal is to extend the ICA process allowing the director to delegate their privileges to a *vice-director*, denoted v, also - and otherwise - operating as a regular employee. While in principle simple, the consideration of delegation introduces several complications.

First, delegation is *dynamic* and its activation/deactivation operates on a time dimension orthogonal to the phases of the ICA process. When activated, delegation should enable the vice-director to perform the second phase also for operations that have originated, and/or whose first phase was even started or completed, before the delegation became active. When deactivated, it should prevent the vice-director from performing the second phase on any operation, even on those that originated, or whose first phase was executed, when the delegation was active. This orthogonal and dynamic lifetime of the delegation requires rethinking the precomputed director tag and phase tag (for its middle layer), statically attached to the operations at their creation.

Second, with the vice-director operating as a regular employee, but also acquiring director's privileges when delegated, care must be taken to ensure the vice-director not to execute the director phase for operations for which the vice-director executed the first phase. With control delegated to the server simply expressed as control on tags (the service provider should remain agnostic with respect to the process itself), the enforcement of this *separation of duties* should be embedded in tags themselves, hence again it requires rethinking the precomputed tag strip (intuitively, treating differently the operations for which the vice-director performed the first phase).

Third, again with the vice-director dynamically acquiring privileges to perform the second phase, care must be taken with respect to potential vulnerabilities of the control, which could be exposed to tampering with the tag strip enabling passing write controls for operations that should not be granted.

In the next section, we redefine tags to address the three challenges above.

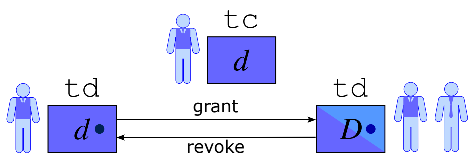


Fig. 3: Evolution of the director tag

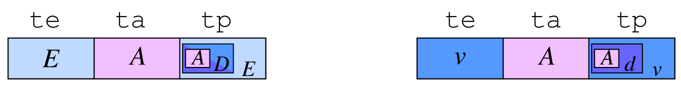
4 Tag Management for Delegation Enforcement

In the following, we describe our solution for supporting director's role delegation addressing the aspects discussed in the previous section. In particular, we illustrate how to support dynamic delegation (Section 4.1), while enforcing separation of duties (Section 4.2), and ensuring integrity of the tag control process and of tag evolution (Section 4.3).

4.1 Dynamic Delegation

As noted, delegation is dynamic and its validity span is orthogonal with respect to the phases of the ICA process. While in the original approach the director tag was static throughout an operation life time and equal for all operations of the same unit, with delegation the director tag needs to change dynamically. This makes the precomputed director tag in the tag strip attached to operations not suitable in presence of delegation. As a matter of fact, maintaining such tag, would require rewriting it for the tag strip of all the existing operations that have not completed the second phase. A further complicating factor is the management of the middle layer of the phase tag associated with operations, which cannot be simply rewritten.

The above observations suggest two requirements: first, the need for a director tag that can be decrypted by both the director and the vice-director (this latter only if delegation is active); second, a detachment of the director tag with respect to operation records. We accommodate them by considering a key, denoted k_D, which can be derived by both the director and vice-director (i.e., $k_d \rightsquigarrow k_D$, and $k_v \rightsquigarrow k_D$). We use key k_D (in contrast to k_d) for the middle layer of the phase tag `tp`, hence enabling its decryption also by the vice-director. We also define, for each unit u, a single director tag u.`td`, which applies to all the operations of the unit. Being detached from operations, this tag can dynamically changed to activate/deactivate delegation as needed. Like in the original model, the tag is a random value encrypted with the unit's director key (i.e., k_d). To activate delegation, the director overwrites the tag using a new random value and key k_D. To deactivate delegation the director overwrites it, again using a new random value, and key k_d. Delegation can be activated and de-activated as needed. To note that activation/deactivation requires not only changing the key with which the tag is encrypted, but also using a new random value to avoid replay attacks. Figure 3 illustrates the change of tag u.`td` when delegation is activated/deactivated, introducing our graphical (double-colored) notation for the



(a) regular employee (b) vice-director

Fig. 4: Structure of the tags at initialization time for operations generated by regular employees (a) and by the vice-director (b)

tag accessible also to the vice-director (i.e., encrypted with key k_D). Again, the bullet in the tag denotes the change of the underlying random value. Note that the director tag can be written only by the director, who is the only subject who can activate/deactivate delegation. Analogously to write operations on reports, write actions on u.`td` are controlled through a tag, denoted u.`tc`, defined at the unit level, and encrypted with key k_d.

Write control (Property 1) needs then to be revised to consider the director tag now associated with the unit (in contrast to the operation). The property is revised as follows, changing the management of write operations on the director report with reference to director tag u.`td` (in contrast to o.`td`).

Property 2 (Write control with delegation). For each subject s, operation o, and report o.`r*` with $* \in \{$`e`,`d`,`a`$\}$:

- `write`(s,o.`r*`) $\in$ W, with `*` $\in \{$`e`,`a`$\}$, iff $\phi(s) \rightsquigarrow \lambda(o.\mathtt{t}*) \wedge \phi(s) \rightsquigarrow \lambda(o.\mathtt{tp})$;
- `write`(s,o.`rd`) $\in$ W iff $\phi(s) \rightsquigarrow \lambda(u.\mathtt{td}) \wedge \phi(s) \rightsquigarrow \lambda(o.\mathtt{tp})$.

4.2 Separation of Duties

With the vice-director also (and otherwise) operating as a regular employee care must be taken to avoid the vice-director to perform both the first (employee) and second (director) phase for an operation as this would violate separation of duties. To ensure this, we remove the vice-director from the set E of employees, treating v as a separate subject. We then consider two different tag strips, one for operations processed by regular employees (i.e., for which a regular employee performed the first phase) and the other for operations processed by the vice-director. The tag strip for operations processed by regular employees is defined as illustrated above, that is, the employee tag `te` (and the external layer of the phase tag `tp`) is encrypted with key k_E and the middle layer of the phase tag is encrypted with key k_D (derivable by both the director and the vice-director, see Figure 4(a)). The tag strip for operations processed by the vice-director (when operating as an employee) have the employee tag `te` (and the external layer of the phase tag `tp`) encrypted with key k_v and the middle layer of the phase tag encrypted with key k_d (see Figure 4(b)). The exclusion of the vice-director from the set E of employees (i.e., $k_v \not\rightsquigarrow k_E$) ensures that the vice-director cannot use the regular employee tag strip. The use of k_d for the middle layer of the phase tag of the vice-director's strip ensures that the vice-director will not be able to

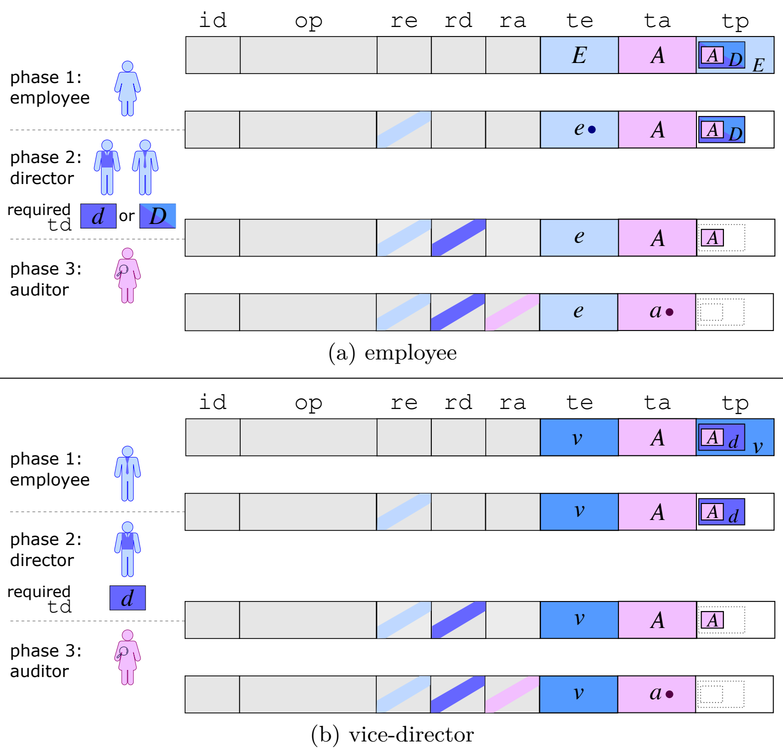


Fig. 5: Evolution of the reports and tags for an operation generated by a regular employee (a) and by the vice-director (b)

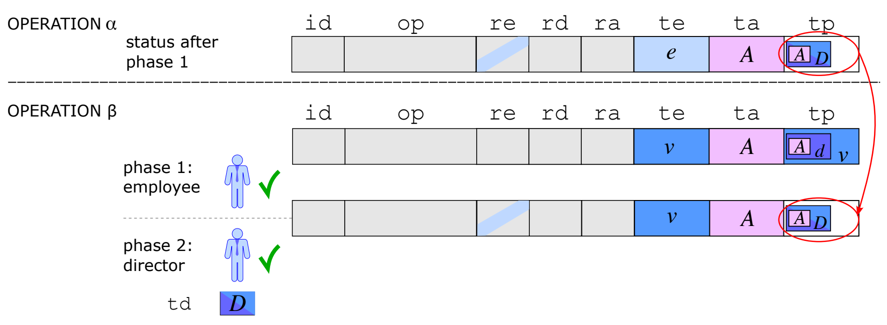
execute the second phase for operations that the vice-director processed, and for which the vice-director executed the first phase. Formally, the generation of the two tag strips is captured by the following property.

Property 3 (Tag strip generation). The tag strip ⟨`te`,`ta`,`tp`⟩ associated with an operation o is computed as follows:

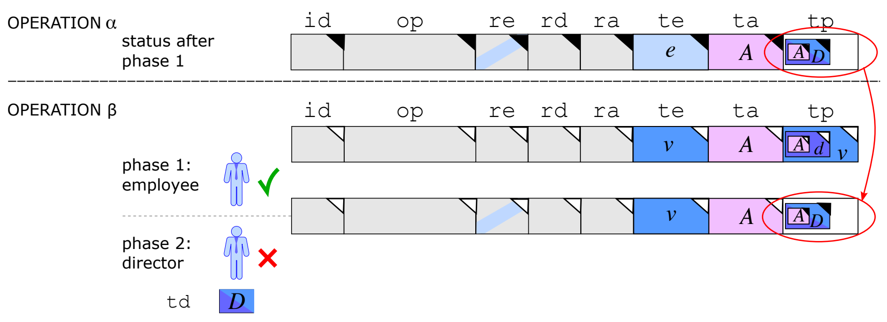
- o.`te`$=$ Enc(σ, k_1);
- o.`ta`$=$ Enc(σ', k_A);
- o.`tp`$=$ Enc(Enc(Enc$(\sigma'', k_A), k_2), k_1)$

where $\sigma \neq \sigma' \neq \sigma''$ are random values, k_1=k_v and k_2=k_d if o is generated by v, or k_1=k_E and k_2=k_D if o is generated by a regular employee.

Figure 5 illustrates the evolution of the tags for operations processed by a regular employee and by the vice-director.



(a) tag mixing



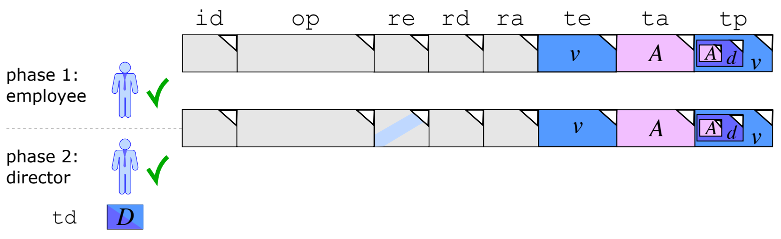
(b) tag mixing - blocked

Fig. 6: An example of tag mixing (`tp` of operation α after phase 1 used for operation β)

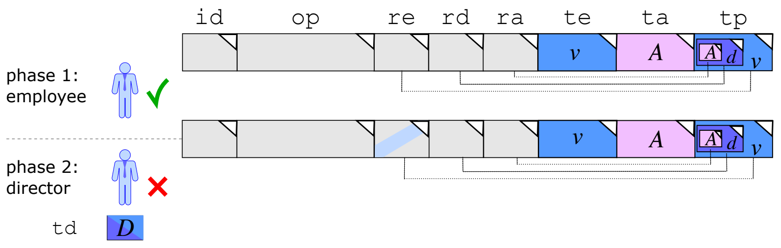
4.3 Ensuring Tag Integrity

Although the use of two different tag strips guarantees the proper execution of the ICA process with delegation without violating the separation of duties principle, the generation of tags must be done with care to avoid possible vulnerabilities. In particular, the mixing of tags from different strips (Figure 6) as well as the phase tag not properly peeled (Figure 7) can cause an unexpected evolution of the ICA process that the cloud provider cannot detect.

– *Tag mixing.* Consider two operations, one generated by a regular employee (α) and another one generated by the vice-director (β), for which the first phase has been completed. Switching the phase tag of the two operations would permit the vice-director, in case delegation is active, to complete the second phase for operation β for which the vice-director also performed the first phase (Figure 6(a)). Indeed, the vice-director would be able to decrypt both `td` and the (middle layer of the) phase tag, both encrypted with key k_D, thus proving to the provider the authorization to write report `rd`. To



(a) no peel



(b) no peel - blocked

Fig. 7: An example of not peeled phase tag

avoid mixing tags of different operations, tags should be tied to operations including the operation identifier (Figure 6(b), where the operation identifier is represented with a triangle of different color in the top left corner of the fields in the operation record).

– *No peel.* Consider an operation generated by the vice-director and assume that, at the end of the first phase, the external layer of the phase tag is not peeled. In case delegation is active, the vice-director would be able to decrypt both td, encrypted with k_D, and tp, encrypted with k_v, thus gaining write access to rd (Figure 7(a)). Note that the vice-director could gain access to rd for such an operation also before the completion of the first phase, or even before its start. To avoid situations where the external layer of the phase tag is used for regulating write access to rd, the layers of tp should include a reference to the phase they regulate (Figure 7(b)).

To tie tags with operations, we add a unique operation identifier within tags. To permit the verification of the correspondence between the phase of the ICA process and the layer of the phase tag, we introduce in each layer a reference to the corresponding phase. Tag strip generation (Property 3) is revised as follows.

Property 4 (Tag strip generation (revised)). The tag strip $\langle$te,ta,tp$\rangle$ associated with an operation o is computed as follows:

– o.te= Enc($\sigma \| o$.id$, k_1$);
– o.ta= Enc($\sigma' \| o$.id$, k_A$);

– $o.\mathtt{tp}$=Enc(Enc(Enc($\sigma''\|o.\mathtt{id}\|\mathtt{a}, k_A)\|o.\mathtt{id}\|\mathtt{d}, k_2)\|o.\mathtt{id}\|\mathtt{e}, k_1$)

where $\sigma \neq \sigma' \neq \sigma''$ are random values, k_1=k_v and k_2=k_d if o is generated by v, or k_1=k_E and k_2=k_D if o is generated by a regular employee.

In the following, we use notation [o.t*].id to refer to the operation identifier in tag o.t*, and [o.tp].ph to refer to the phase reported in the phase tag. Note that, even including the operation (and phase) identifier, tags are always generated starting from a random value since otherwise anyone could prove ability to decrypt tags, being the operation identifier publicly available. The random values used to generate tag strips are all different, that is, tags of the same or of different operations are generated from different random values.

Write control (Property 2) is revised as follows to consider operation identifiers within tags and phase reference in the layers of the phase tag.

Property 5 (Write control with delegation (revised)). For each subject s, operation o processed at unit u, and report o.r* with * ∈ {e,d,a}:

- write(s,o.r*) ∈ W, with * ∈ {e,a}, iff $\phi(s)\rightsquigarrow\lambda(o.\mathtt{t*}) \wedge \phi(s)\rightsquigarrow\lambda(o.\mathtt{tp}) \wedge$ o.id=[o.t*].id=[o.tp].id ∧ [o.tp].ph=*;
- write(s,o.rd) ∈ W iff $\phi(s)\rightsquigarrow\lambda(u.\mathtt{td}) \wedge \phi(s)\rightsquigarrow\lambda(o.\mathtt{tp}) \wedge$ o.id=[o.tp].id ∧ [o.tp].ph=d.

Analogously to the write control on reports, also the evolution of tags is regulated (the evolution of the ICA process requires report and tag updates). The rules governing tag updates are similar to those regulating report updates. A subject is authorized to write a report tag t* (with * ∈ {e,a}) if the subject would also be authorized to write the corresponding report r* (i.e., the first condition in Property 5 holds also for tags te and ta). Only the director is authorized to write td. A subject is authorized to write the phase tag tp if the subject can decrypt the phase tag, as well as the report tag corresponding to the exposed layer of the phase tag, and the operation identifier matches with the one of the phase tag and of the provided report tag (e.g., the subject can decrypt tp and te, [o.tp].ph=e, and o.id=[o.te].id=[o.tp].id). Note that, checking the ability of a subject to decrypt a report tag besides the phase tag, prevents subjects who can decrypt the phase tag to terminate a phase when not responsible for it. Write control on tags is formally captured by the following property.

Property 6 (Write control on tags). For each subject s, operation o processed at unit u, and tag o.t* with $*$ ∈ {e,a,p} and tag u.td:

- write(s,o.t*) ∈ W, with * ∈ {e,a}, iff $\phi(s)\rightsquigarrow\lambda(o.\mathtt{t}*) \wedge \phi(s)\rightsquigarrow\lambda(o.\mathtt{tp}) \wedge$ o.id=[o.t*].id=[o.tp].id ∧ [o.tp].ph=*;
- write(s,u.td) ∈ W, iff $\phi(s)=\lambda(u.\mathtt{tc})$;
- write(s,o.tp) ∈ W iff $\phi(s)\rightsquigarrow\lambda(o.\mathtt{tp})$ ∧ o.id=[o.tp].id ∧ (($\phi(s)\rightsquigarrow\lambda(o.\mathtt{te})$ ∧ [o.tp].ph=e ∧ o.id=[o.te].id) ∨ ($\phi(s)\rightsquigarrow\lambda(u.\mathtt{td})$ ∧ [o.tp].ph=d) ∨ ($\phi(s)\rightsquigarrow\lambda(o.\mathtt{ta})$ ∧ [o.tp].ph=a ∧ o.id=[o.ta].id)).

We conclude this section observing that the tag strip ⟨te,ta,tp⟩ associated with an operation does not need to be created when the operation is generated,

```
**Delegation**($u$,*action*) /* client-side; perform action on director's role at unit $u$ * /
1: $\sigma_d$=Dec($u$.tc, $k_d$) /* decryption of tag regulating write operations over $u$.td */
2: if action==activate then /* activate delegation */
3:   key=$k_d$, new_key=$k_D$
4: else /* deactivate delegation */
5:   key=$k_D$; new_key=$k_d$
6: $\sigma$=Dec($u$.td, key)
7: randomly generate $\sigma'$
8: new_tag=Enc($\sigma'$, new_key)
9: Write_TD($u$, $\sigma$, $\sigma_d$, new_tag, new_key)

Write_TD($u$, $\sigma$, $\sigma_d$, new_tag, new_key) /* provider-side; check $\sigma$ and $\sigma_d$ to verify if $u$.td */
1: if $\sigma$==Dec($u$.td, $\lambda$($u$.td)) AND /* can be overwritten with new_tag encrypted with new_key */
   $\sigma_d$==Dec($u$.tc, $\lambda$($u$.tc)) then
2:   $u$.td=new_tag
3:   $\lambda$($u$.td)=new_key
```

Fig. 8: Procedures for activating and deactivating delegation

but it can be precomputed in advance as discussed in [4]. The only difference in our approach compared to [4] is that, since our tags include the operation identifier, we precompute a set of tag strips for each unit, together with operation identifiers. When a new operation is generated, it is associated with an identifier and the appropriate tag strip depending on who has generated the operation (a regular employee or the vice-director).

5 Management of Delegation and Write Operations

In this section, we describe the pseudocode of the activation and deactivation of delegation (Section 5.1) and of write operations on reports (Section 5.2).

5.1 Delegation

Figure 8 illustrates the pseudocode implementing the activation and deactivation of delegation. Procedure **Delegation** is invoked by the director d of a unit u and takes as input the unit u of the director and the *action* (activation or deactivation) that the director wishes to perform. We assume that before executing the procedure, the director tag has been correctly generated, meaning that the tag was initially encrypted with key k_d. The procedure first decrypts the tag u.tc governing write operations over u.td (line 1). It then determines the key (variable *key*) protecting the director tag and the new key (variable *new_key*) that will be used for encrypting the director tag (lines 2-5). The value of these keys depends on the action to be enforced. In case of activation, the tag must be decrypted with *key*=k_d (i.e., the key associated with the subject executing the procedure) and re-encrypted with *new_key*=k_D (i.e., the key shared with the vice-director and that the subject calling the procedure should derive starting from their own key); *key*=k_D and *new_key*=k_d, otherwise. The procedure then decrypts the director tag u.`td` with *key* (line 6), generates a new random value σ' (line 7), and encrypts it with *new_key* (line 8). The procedure finally invokes **Write_TD** operating at the provider side (line 9).

Procedure **Write_TD** takes as input the unit u (i.e., the unit of director d), values σ and σ_d obtained from procedure **Delegation** through the decryption of tags u.`td` and u.`tc`, respectively, the new director tag *new_tag* computed by procedure **Delegation**, and the new key *new_key* used for encrypting *new_tag*. **Write_TD** verifies whether σ and σ_d match the decryption of u.`tc` and u.`td`, respectively (line 1). If this is the case, the cloud provider concludes that procedure **Delegation** has been called by the director of unit u. Procedure **Write_TD** can then overwrite the director tag u.`td` with *new_tag* (line 2). The procedure also updates the identifier of the encryption key now protecting u.`td` (line 3). We note that the checks on σ and σ_d (line 1) guarantee that only the director of a unit can activate delegation since the tag regulating write operations on the director tag (i.e., tag u.`tc`) is encrypted with a key known to the director only.

5.2 Write Reports and Tags

Figure 9 illustrates the pseudocode implementing the evolution of an ICA phase. When a subject s needs to perform an ICA phase over an operation, s invokes procedure **ICA_Phase**, which takes as input the operation identifier *id*, the unit u of the operation, and the phase *phase* of the ICA process. Depending on the value of variable *phase*, the procedure identifies the report `r` to be generated/updated, and the corresponding report tag and decrypts it (lines 2-5). The procedure then decrypts the phase tag `tp`, obtaining triple (σ_p,id_p,p) (line 6). Note that a subject s can decrypt a tag if and only if the subject can derive the key used for encrypting the tag. Otherwise, the decryption operation does not produce a meaningful result.

For the employee and auditor phases, if the operation identifier reported in the decrypted report tag and phase tag matches the operation identifier o.`id`, the procedure generates a new random value σ', computes a new report tag *new_tag* concatenating σ' with o.`id` and encrypting it with the key of the employee/auditor who started the phase, and writes it in the operation record by invoking procedure **Write_Tag** executed at the provider (lines 7-11). Procedure **ICA_Phase** then checks if the operation identifiers in the phase tag id_p and in the report tag id_r correspond to the input *id* and if the input *phase* corresponds to the phase in the phase tag. If this is the case, the considered tags are those associated with operation o and therefore procedure **ICA_Phase** proceeds with the generation/update of the report of interest and writes it in the operation record by invoking procedure **Write_Report** executed at the provider (lines 12-15). Finally, procedure **ICA_Phase** completes the ICA phase by invoking procedure **Peel_Phase_Tag**, which operates at the provider and removes the exposed encryption layer of the phase tag.

Procedure **Write_Tag** enforces Property 6 and is executed by the provider. It verifies whether subject s is authorized to write a report tag (i.e., o.`te` or o.`ta`)

```
**ICA_Phase**($id$, $u$, $phase$) /* client-side; write report of $phase$ for operation $id$ at unit $u$ */
 1: let $o$ in $O$ s.t. $o$.id=$id$
 2: case $phase$ of
 3:   e: r=$o$.re, $(\sigma_r, id_r)$=Dec($o$.te,$\lambda$($o$.te)) /* $\phi(s)\rightsquigarrow\lambda(o.\mathtt{te})$ */
 4:   d: r=$o$.rd, $\sigma_r$=Dec($u$.td,$\lambda$($u$.td)), $id_r=id$ /* $\phi(s)\rightsquigarrow\lambda(u.\mathtt{td})$ */
 5:   a: r=$o$.ra, $(\sigma_r, id_r)$=Dec($o$.ta,$\lambda$($o$.ta)) /* $\phi(s)\rightsquigarrow\lambda(o.\mathtt{ta})$ */
 6: $(\sigma_p, id_p, p)$=Dec($o$.tp,$\lambda$($o$.tp)) /* $\phi(s)\rightsquigarrow\lambda(o.\mathtt{tp})$ */
 7: if ($p$==$phase$==e OR $p$==$phase$==a) AND $id_r$==$id_p$==$o$.id then /* start ICA $phase$ */
 8:   randomly generate $\sigma'$
 9:   $new\_tag$=Enc($\sigma'\|o$.id,$k_s$) /* with $s$ the invoking subject */
10:   $new\_key=k_s$
11:   Write_Tag($id$, $\sigma_r$, $\sigma_p$, $new\_tag$, $new\_key$, $phase$)
12: if $id_r$==$id_p$==$o$.id AND $p$==$phase$ then /* generate/modify and write the report */
13:   r=Decrypt $o$.r
14:   r=Update and encrypt r
15:   Write_Report($id$, $u$, $\sigma_r$, $\sigma_p$, r, $phase$)
16:   Peel_Phase_Tag($id$, $u$, $\sigma_r$, $\sigma_p$, $phase$) /* finalize ICA $phase$ */

**Write_Tag**($id$, $\sigma_r$, $\sigma_p$, $t$, $k$, $phase$) /* provider-side; check $\sigma_r$ and $\sigma_p$ to verify if */
 1: let $o$ in $O$ s.t. $o$.id=$id$ /* tag of $phase$ for operation $id$ can be set to $t$ encrypted with $k$ */
 2: case $phase$ of
 3:   e: t=$o$.te
 5:   a: t=$o$.ta
 6: if $(\sigma_r, id)$==Dec(t,$\lambda$(t)) AND $(\sigma_p, id, phase)$==Dec($o$.tp,$\lambda$($o$.tp)) then
 7:   t=$t$; $\lambda$(t)=$k$

**Write_Report**($id$, $u$, $\sigma_r$, $\sigma_p$, $r$, $phase$) /* provider-side; check $\sigma_r$ and $\sigma_p$ to verify if */
 1: let $o$ in $O$ s.t. $o$.id=$id$ /* report of $phase$ for operation $id$ at unit $u$ can be set to $r$ */
 2: case $phase$ of
 3:   e: r=$o$.re, t=$o$.te
 4:   d: r=$o$.rd, t=$u$.td
 5:   a: r=$o$.ra, t=$o$.ta
 6: if (($phase$==d AND $\sigma_r$==Dec(t,$\lambda$(t))) OR $(\sigma_r, id)$==Dec(t,$\lambda$(t))) AND
    $(\sigma_p, id, phase)$==Dec($o$.tp,$\lambda$($o$.tp)) then r=$r$

**Peel_Phase_Tag**($id$, $u$, $\sigma_r$, $\sigma_p$, $phase$) /* provider-side; check $\sigma_r$ and $\sigma_p$ to verify if */
 1: let $o$ in $O$ s.t. $o$.id=$id$ /* phase tag at $phase$ of operation $id$ at unit $u$ can be peeled */
 2: case $phase$ of
 3:   e: t=$o$.te
 4:     if $\lambda$($o$.te)==$k_{v_u}$ then $key$=$k_{d_u}$ /* keys of vice-director ($v_u$) and director ($k_{d_u}$) of unit $u$ */
 5:     else $key$=$k_{D_u}$ /* key shared between director and vice-director of unit $u$ */
 6:   d: t=$u$.td, $key$=$k_A$
 7:   a: t=$o$.ta, $key$=NULL
 8: if (($phase$==d AND $\sigma_r$==Dec(t,$\lambda$(t))) OR $(\sigma_r, id)$==Dec(t,$\lambda$(t))) AND
    $(\sigma_p, id, phase)$==Dec($o$.tp,$\lambda$($o$.tp)) then $o$.tp=$\sigma_p$; $\lambda$($o$.tp)=$key$
```

Fig. 9: Execution of a ICA phase

by checking whether: 1) the input value σ_p corresponds to the decryption of the phase tag (i.e., subject s can derive $\lambda(o.\texttt{tp})$), 2) the input value σ_r corresponds to the decryption of the report tag (i.e., subject s can derive $\lambda(o.\texttt{t}*)$), 3) the input operation identifier *id* matches the identifier reported in the report and phase tags, and 4) the input phase corresponds to the phase reported in the phase tag. If these checks succeed, the provider can conclude that s is authorized to write the report tag, and updates it with the value received as input.

Analogously, procedure **Write_Report** enforces Property 5 and is executed by the provider. It verifies whether subject s is authorized to write a report (i.e., $o.\texttt{re}$ or $o.\texttt{rd}$ or $o.\texttt{ra}$) by checking whether: 1) the input value σ_p corresponds to the decryption of the phase tag (i.e., subject s can derive $\lambda(o.\texttt{tp})$), 2) the input value σ_r corresponds to the decryption of the report or director tag (i.e., subject s can derive $\lambda(o.\texttt{te})$ or $\lambda(u.\texttt{td})$ or $\lambda(o.\texttt{ta})$); 3) the (employee or auditor) report tag and the phase tag are those associated with the operation of interest (i.e., the input operation identifier *id* matches with the identifier in the tags), and 4) the input phase corresponds to the phase reported in the phase tag. If these checks succeed, the provider can conclude that s is authorized to write the report, and modifies it according to the input value.

Procedure **Peel_Phase_Tag** takes the same input as procedure **Write_Report** and performs the same checks. If the checks succeed, **Peel_Phase_Tag** removes the exposed layer of the phase tag.

6 Related Work

The adoption of cloud services for data storage and management provides numerous advantages but also introduces several issues related to, for example, the reliability of cloud providers and the lack of control of data owners over their data and processing (e.g., [3, 6, 9, 10, 15, 17]). The problem of protecting data and computations (confidentiality and integrity) when moving to the cloud has been widely studied. Solutions protecting the confidentiality of outsourced data are often based on owner-side encryption (e.g., [7]), thus preventing exposure of sensitive information if the cloud provider is compromised. Owner-side encryption, however, rules out any processing of data. Many efforts have been then dedicated to the design of approaches for supporting computations over encrypted data (e.g., [5, 7, 11, 13]). With respect to integrity, solutions have been proposed for verifying not only that data are correctly stored at the cloud provider but also the integrity of data processing results (e.g., [18]).

A line of research related to our work focuses on the problem of enforcing access control on outsourced data. Approaches addressing this problem either rely on attribute-based encryption (ABE) (e.g., [16, 19]), or combine selective encryption and key derivation strategies (e.g., [3, 12]) to translate read access privileges into the knowledge of the keys necessary to decrypt data. ABE is a public key encryption schema that can be combined with ABS (Attribute-Based Signature) for regulating write actions over resources (e.g., [8, 14]). While effective, solutions relying on ABE are less efficient than our proposal due to their adoption of asymmetric encryption. Our proposal is inspired by approaches leveraging selective encryption for access control enforcement. Selective encryption uses symmetric encryption and enforces access control policies by properly regulating the keys to be used for resource encryption and to be distributed to users. Key derivation strategies (e.g., [1]) enable users to derive, from the distributed keys, the ones used for encryption. Selective encryption approaches have also been enhanced to enforce selective write privileges (e.g., [2]). While similar to our proposal, these techniques cannot be directly used for enforcing the ICA process, because of the peculiarities of the considered scenario and of the intrinsically dynamic nature of authorizations while the process evolves.

The problem of moving the ICA process to the cloud has been first addressed in [4]. However, this proposal does not consider the potential block in the operativity of a unit caused by the absence of the director responsible for the second phase of the ICA process of all operations of the unit. We have then enhanced this solution to enable the delegation of the director privileges, while preventing the delegated subject to perform more than one phase of the ICA process.

7 Conclusions

We addressed the problem of enforcing corporate governance internal control and audit functions while relying on cloud-based services for their execution. Extending previous work, we considered in particular the support of delegation, enabling directors to dynamically and temporarily delegate their privileges to vice-directors. Support of dynamic delegation required rethinking the (predefined and static) encrypted tags attached to operations. Our solution provides support of delegation while ensuring separation of duties and correctness of the control against possible misbehavior and tag tampering. Our work leaves room for extensions, including the support of alternative delegation (e.g., non-predefined vice-director, enforcement of two-person-rules for acquiring director's privileges), and consideration of additional functionalities of governance control (e.g., preventing subjects to perform employee, director, or audit control on operations for which they may be in conflict of interest).

Acknowledgements. This work was supported in part by the EC under projects GLACIATION (101070141) and EdgeAI (101097300), by the Italian MUR under PRIN project POLAR (2022LA8XBH), and by project SERICS (PE00000014) under the MUR NRRP funded by the EU - NGEU. Project EdgeAI is supported by the Chips Joint Undertaking and its members including top-up funding by Austria, Belgium, France, Greece, Italy, Latvia, Netherlands, and Norway under grant agreement No. 101097300. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union, the Chips Joint Undertaking, or the Italian MUR. Neither the European Union, nor the granting authority, nor Italian MUR can be held responsible for them.

References

1. Atallah, M., Blanton, M., Fazio, N., Frikken, K.: Dynamic and efficient key management for access hierarchies. ACM TISSEC **12**(3), 18:1–18:43 (Jan 2009)
2. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., Samarati, P.: Enforcing dynamic write privileges in data outsourcing. COSE **39**, 47–63 (Nov 2013)
3. 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 (Apr 2010)
4. De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Samarati, P.: Enforcing corporate governance controls with cloud-based services. IEEE TSC **17**(6), 3583–3596 (Nov-Dec 2024)
5. Ding, X., Wang, Z., Zhou, P., Choo, K.K.R., Jin, H.: Efficient and privacy-preserving multi-party skyline queries over encrypted data. IEEE TIFS **16**, 4589–4604 (Aug 2021)
6. Gritzalis, S., Yannacopoulos, A., Lambrinoudakis, C., Hatzopoulos, P., Katsikas, S.: A probabilistic model for optimal insurance contracts against security risks and privacy violation in IT outsourcing environments. IJIS **6**, 197–211 (Jan 2007)
7. Hacigümüş, H., Iyer, B., Mehrotra, S., Li, C.: Executing SQL over encrypted data in the database-service-provider model. In: Proc. of SIGMOD. Madison, WI, USA (Jun 2002)
8. Huang, Q., Yang, Y., Shen, M.: Secure and efficient data collaboration with hierarchical attribute-based encryption in cloud computing. Future Generation Computer Systems **72**, 239–249 (Jul 2017)
9. Jhawar, R., Piuri, V., Samarati, P.: Supporting security requirements for resource management in cloud computing. In: Proc. of CSE. Paphos, Cyprus (December 2012)
10. Jhawar, R., Piuri, V., Santambrogio, M.: A comprehensive conceptual system-level approach to fault tolerance in cloud computing. In: Proc. of SysCon. Vancouver, BC, Canada (March 2012)
11. Li, F., Ma, J., Miao, Y., Liu, X., Ning, J., Deng, R.H.: A survey on searchable symmetric encryption. ACM CSUR **56**(5), 119:1 – 119:42 (May 2024)
12. Miklau, G., Suciu, D.: Controlling access to published data using cryptography. In: Proc. of VLDB. Berlin, Germany (Sep 2003)
13. Poh, G., Chin, J., Yau, W., Choo, K.K.R., Mohamad, M.: Searchable symmetric encryption: Designs and challenges. ACM CSUR **50**(3), 40:1–40:37 (2017)
14. Ruj, S., Stojmenovic, M., Nayak, A.: Privacy preserving access control with authentication for securing data in clouds. In: Proc. of CCGrid. Ottawa, Canada (May 2012)
15. Xie, S., Mohammady, M., Wang, H., Wang, L., Vaidya, J., Hong, Y.: A generalized framework for preserving both privacy and utility in data outsourcing. IEEE TKDE **35**(1), 1–15 (Jan 2023)
16. Xu, S., Ning, J., Huang, X., Li, Y., Xu, G.: Untouchable once revoking: A practical and secure dynamic EHR sharing system via cloud. IEEE TDSC **19**(6), 3759–3772 (Nov-Dec 2022)
17. Zahid, M., Shafiq, B., Vaidya, J., Afzal, A., Shamail, S.: Collaborative business process fault resolution in the services cloud. IEEE TSC **16**(1), 162–176 (Jan-Feb 2023)

18. Zhang, B., Dong, B., Wang, W.: Integrity authentication for SQL query evaluation on outsourced databases: A survey. IEEE TKDE **33**(4), 1601–1618 (Apr 2021)
19. Zhang, Y., Deng, R., Xu, S., Sun, J., Li, Q., Zheng, D.: Attribute-based encryption for cloud computing access control: A survey. ACM CSUR **53**(4), 83:1–83:41 (Jul 2021)

Correctness

We first show that **Delegation** procedure updates the director tag in accordance with the status of delegation.

Lemma 1. *Procedure* **Delegation** *in Figure 8 guarantees that for each unit u,* $\phi(s) \rightsquigarrow \lambda(u.\texttt{td})$ *iff* $s{=}d$*, or* $s{=}v$ *and delegation is active.*

Proof. We assume that the director tag is correctly generated and encrypted when procedure **Delegation** is called and that the caller of the procedure is the director d of unit u. We now distinguish two cases, depending on the action (activate or deactivate) input to the procedure.

– **Delegation**(u,activate). The procedure first decrypts the control tag $u.\texttt{tc}$, obtaining value σ_d. The director tag is encrypted with k_d that only d can derive. The procedure can then decrypt the director tag (obtaining value σ), generate a new random value, and re-encrypt the new random value with k_D (*new_tag*) that both d and v can derive. The verification of whether the caller of **Delegation** is authorized to overwrite the director tag with *new_tag* (i.e., to activate delegation) is verified by the provider through procedure **Write_TD**, which checks whether both σ_d and σ match with the decryption of $\lambda(u.\texttt{tc})$ and $\lambda(u.\texttt{td})$, respectively. If this is the case, **Write_TD** activates delegation by overwriting $u.\texttt{td}$ with *new_tag* and assigning k_D to $\lambda(u.\texttt{td})$. After the activation of delegation, $\phi(s) \rightsquigarrow \lambda(u.\texttt{td})$, with $s{\in}\{d{,}v\}$.
– **Delegation**(u,deactivate). The procedure first decrypts the control tag $u.\texttt{tc}$, obtaining value σ_d. The director tag is encrypted with k_D that d and v can derive. The procedure can then decrypt the director tag with k_D, generate a new value, and re-encrypt the new value with k_d that only subject d can derive. Again, the verification of whether the caller of **Delegation** is authorized to overwrite the director tag with *new_tag* (i.e., to deactivate delegation) is verified by the provider through procedure **Write_TD**. If such a check succeeds, **Write_TD** overwrites $u.\texttt{td}$ with *new_tag*, and assigns k_d to $\lambda(u.\texttt{td})$. As a consequence, delegation is deactivated and $\phi(s) \rightsquigarrow \lambda(u.\texttt{td})$, with $s{=}d$. □

We now show that **Peel_Phase_Tag** procedure correctly decrypts the phase tag of an operation.

Lemma 2. *Procedure* **Peel_Phase_Tag** *in Figure 9 guarantees that for each object o, phase tag* $o.\texttt{tp}$ *is correctly decrypted.*

Proof. The procedure takes as input the *id* of an operation, the unit u where operation *id* has been processed, two secret values σ_r and σ_p, and the current ICA *phase*. Depending on the current *phase*, the procedure determines the key needed for decrypting the next layer of the phase tag. The procedure then checks whether the decryption of o.`tp` produces a triple that corresponds to (σ_p,*id*,*phase*), and the input value σ_r corresponds to the decryption of the director tag (*phase*=`d`) or the input *id* and σ_r correspond to the decryption of the report tag (*phase*=`e` or *phase*=`a`). If the control succeeds, the procedure assigns σ_p to o.`tp`, meaning that the current layer of the phase tag is correctly removed. □

We are now ready to prove that the procedures in Figure 9 satisfy Property 5.

Theorem 1 (Correct enforcement of write control). *Procedure* **ICA_Phase** *guarantees that for each subject s and operation o, Property 5 is satisfied.*

Proof. We assume that before starting the ICA process, the tags associated with the operations have been correctly generated and processed. We distinguish two cases: *1)* the subject invoking the procedure is an employee or an auditor and the phase is `e` or `a`, and *2)* the subject invoking the procedure is the director or the vice-director and the phase is `d`.

– *Case 1.* The procedure decrypts both the report tag corresponding to the input value *phase* (i.e., `te` if *phase*=`e` or `ta` if *phase*=`a`) obtaining pair (σ_r,*id*), and the phase tag obtaining triple (σ_p,*id*,p). If there is a match among the identifier provided as input and the ones obtained from the decryption of the report and phase tags and the phase tag exposes the encryption layer corresponding to the input *phase*, the procedure calls **Write_Report**. Procedure **Write_Report** checks whether the report and phase tags have been correctly decrypted (input σ_r and σ_p), the operation identifier (input *id*) matches the identifier reported in the tags, and the input phase *phase* corresponds to the phase reported in the phase tag. The check performed by **Write_Report** succeeds only if: 1) $\phi(s)\rightsquigarrow\lambda(o.\texttt{t*})$ (with `t*`=`te` or `t*`=`ta`) because only in this case σ_r can correspond to the value that the cloud provider obtains with the decryption of the report tag; 2) $\phi(s)\rightsquigarrow\lambda(o.\texttt{tp})$ because only in this way σ_p can correspond to the value that the cloud provider obtains with the decryption of the phase tag; 3) the input *id* corresponds to the identifiers that the cloud provider obtains from the decryption of the report and phase tags; 4) the input *phase* corresponds to the phase that the cloud provider obtains from the decryption of the phase tags. The report is then written only if Property 5 is satisfied.
– *Case 2.* For the operation o the employee phase has been already executed as the requesting input *phase* is `d`. Procedure **Peel_Phase_Tag** has then removed the external layer (Lemma 2) of the phase tag that now exposes the layer corresponding to the director phase. The **ICA_Phase** procedure decrypts both the director tag obtaining value σ_r, and the phase tag obtaining triple (σ_p,*id*,p). If the operation identifier (input *id*) matches the identifier

reported in the phase tag, and the input phase *phase* corresponds to the phase reported in the phase tag, the procedure calls **Write_Report** that, as before, checks whether the director and phase tags have been correctly decrypted (input σ_r and σ_p), the operation identifier (input *id*) matches the identifier reported in the phase tag, and the input phase *phase* corresponds to the phase reported in the phase tag. This check succeeds only if: 1) $\phi(s) \rightsquigarrow \lambda(o.\texttt{td})$ meaning that s is the director or, if the delegation is active (Lemma 1), the vice-director; 2) $\phi(s) \rightsquigarrow \lambda(o.\texttt{tp})$ because only in this way σ_p can correspond to the value that the cloud provider obtains with the decryption of the phase tag; 3) *id* corresponds to the identifiers that the cloud provider obtains from the decryption of the director and phase tags; 4) *phase* corresponds to the phase that the cloud provider obtains from the decryption of the phase tags. The report is then written only if Property 5 is satisfied.