Project title: Enforceable Security in the Cloud to Uphold Data Ownership
Project acronym: ESCUDO-CLOUD
Funding scheme: H2020-ICT-2014
Topic: ICT-07-2014
Project duration: January 2015 – December 2017

D3.1
Report on techniques for selective access

Editors: Sara Foresti (UNIMI)
Giovanni Livraga (UNIMI)
Reviewers: Nikola Knežević (IBM)
Luis Alcántara García (WT)

Abstract

Today, users placing their data in the cloud are more and more relying on services and facilities offered by cloud service providers for easily sharing resources with others. Data sharing is however typically selective, that is, the owner of each resource may wish to enforce restrictions on who can access her resources in the cloud. Since the enforcement of such restrictions can be delegated neither to the owner (for practical and efficiency reasons) nor to the cloud provider (for security reasons), the data themselves need to self-enforce access control rules. To this aim, in this deliverable we illustrate an approach for selective access that is based on selective encryption, which consists in adopting different keys to protect different pieces of information and in distributing keys to users according to their access privileges. We then focus on the supply chain scenario, and describe the use of RFID tags for authentication and of selective encryption for access control enforcement.

Type	Identifier	Dissemination	Date
Deliverable	D3.1	Public	2015.12.31

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644579. This work was supported in part by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract No 150087. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission or the Swiss Government.

ESCUDO-CLOUD Consortium

1.	Università degli Studi di Milano	UNIMI	Italy
2.	British Telecom	BT	United Kingdom
3.	EMC Corporation	EMC	Ireland
4.	IBM Research GmbH	IBM	Switzerland
5.	SAP SE	SAP	Germany
6.	Technische Universität Darmstadt	TUD	Germany
7.	Università degli Studi di Bergamo	UNIBG	Italy
8.	Wellness Telecom	WT	Spain

Versions

Version	Date	Description
0.1	2015.11.23	Initial Release
0.2	2015.12.14	Second Release
1.0	2015.12.31	Final Release

List of Contributors

This document contains contributions from different ESCUDO-CLOUD partners. Contributors for the chapters of this deliverable are presented in the following table.

Chapter	Author(s)
Executive Summary	Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI)
Chapter 1: Introduction	Sara Foresti (UNIMI)
Chapter 2: Selective access	Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Giovanni Livraga (UNIMI), Stefano Paraboschi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 3: Selective access for supply chain management	Florian Kerschbaum (SAP)
Chapter 4: Conclusions	Sara Foresti (UNIMI)

Contents

List of Figures

Executive Summary

The wide availability of cloud service providers offering a variety of services at convenient prices, and the growing amount of information generated every day are encouraging users and companies to move their data to the cloud. Since cloud providers are often considered honest-but-curious (i.e., trusted to correctly manage the data they store but not trusted to access their content), users can wrap their data in a protective encryption layer before storing them in the cloud. If the encryption key is kept secret and communicated to authorized users only, encryption protects data confidentiality against both unauthorized users and curious providers. One of the reasons why users move their data to the cloud is however represented by the availability of facilities for easily sharing their data with others. Such a sharing is typically selective (i.e., each user should be able to access only the resources for which she is authorized by the data owner), and neither the data owner nor the cloud provider should be in charge of enforcing access restrictions. In fact, the enforcement of selective access by the data would require her to mediate every access request, thus nullifying the advantages of moving to the cloud. On the other hand, the cloud provider, being not fully trusted, cannot be authorized for this task.

The first goal of this deliverable is to illustrate an approach able to guarantee that outsourced data self-enforce access restrictions. This technique relies on selective encryption, which consists in encrypting different pieces of information with different encryption keys according to the set of users who can access them. These encryption keys are then distributed to users in such a way that each user can decrypt all and only the data that she is authorized to access. The authorization policy defined by the data owner is then translated into an equivalent encryption policy, regulating key distribution to users and resources (for their encryption). Although effective, selective encryption requires data re-encryption every time a user is granted or revoked access to a piece of information. This deliverable presents an approach able to partially delegate to the cloud provider the burden of managing updates to the access policy. The deliverable also presents an approach, based on selective encryption, for enforcing write access restrictions (which the data owner may want to grant to selected users), supporting also updates in the write authorization policy.

The second goal of this deliverable is to apply selective encryption techniques to the supply chain scenario, which characterizes ESCUDO-CLOUD Use Case 2. In this scenario, a set of mobile physical objects are handled by a set of players, which collect and store in the cloud information about the objects themselves. Clearly, not all the players in the system are authorized to access all the data about all the objects, but only a subset of them (e.g., the players that handled an object can access its data). To properly enforce access control over the data collected in a supply chain, this deliverable presents a solution that adopts RFID tags to authenticate players (and to determine whether a player handled a given object), and selective encryption to enforce access restrictions over the data in the cloud.

1. Introduction

In this chapter, we illustrate the reference scenario and the problem addressed in the remainder of this deliverable.

1.1 Scenario and problem definition

Moving data storage and management to the cloud clearly provides data owners and final users with considerable advantages related, for instance, to lower costs, higher availability, and larger elasticity that are offered by the growing cloud market. A major obstacle refraining from the wide adoption of cloud services is represented by the loss of control over sensitive data. In fact, outsourced data are no more under the direct control of their owner as they are stored at a honest-but-curious provider. A common approach adopted to guarantee confidentiality of the data content consists in encrypting the data before outsourcing them. Since the encryption key is communicated to authorized users only, neither unauthorized users nor the cloud provider can access the sensitive data content.

Although user-side encryption is effective in protecting data confidentiality to the eyes of non-authorized users and curious providers, it naturally provides complete plaintext access to every user authorized for the system. In a wide cloud scenario, characterized by the presence of a variety of users, data owners may need to enforce a specific *access control policy*. An access control policy is a set of rules defined by the data owner specifying who can access what piece of her data in the cloud. If a single key is used to protect the whole data collection, selective data sharing can be enforced only by the data owner (who would need to filter every access request) or by the cloud provider (who would need to know the access control policy). However, neither of these parties can enforce selective access to the data in the cloud, for efficiency and privacy reasons, respectively.

The reference scenario considered in this deliverable is illustrated in Figure 1.1 and is characterized by the presence of one data owner, who stores her data at one cloud provider and needs to selectively share her data with other users. Users can be authorized to read or write (or both) the owner's data according to the data owner's access control policy, which may vary over time.

In this deliverable, we present a solution able to enforce selective access privileges over outsourced data without requiring the intervention of the data owner or of the cloud provider. This solution is based on selective encryption, that is, on the adoption of different encryption keys to protect different portions of the data. If encryption keys are properly distributed to users, we can guarantee that each user can decrypt all and only the data she is authorized to read. Since the data owner may be willing to allow trusted users to also modify her data, this basic approach is extended to support the enforcement of write privileges. A challenge that needs to be addressed when relying on selective encryption for access control enforcement is the management of policy updates. Indeed, a change in the set of users who can access a piece of information would trivially

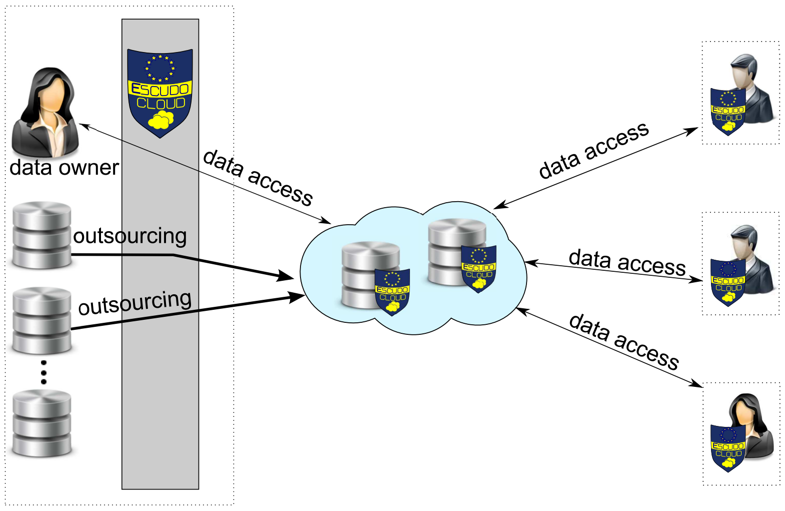


Figure 1.1: Reference scenario

require re-encryption of the data with a new key. To avoid such an expensive operation, this deliverable illustrates a solution that relies on two encryption layers, one managed by the data owner and one managed by the cloud provider, to partially delegate policy updates to the latter.

Selective encryption approaches for access control enforcement can be effectively applied in the supply chain scenario (ESCUDO-CLOUD Use Case 2), which is characterized by multiple players that store data about the objects they handle in a database stored in the cloud. In fact, data collected about the objects processed by a supply chain can be accessed only by a subset of the parties in the system, depending on their role in the supply chain and on the specific policy regulating access to the dataset. This deliverable first presents a solution for authenticating (authorized) players within a supply chain through RFID tags. Then, it illustrates the adoption of selective encryption for enforcing access control over the data collected in the supply chain management and stored in the cloud repository.

1.2 Outline

The remainder of this document is organized as follows.

- Chapter 2 illustrates an approach based on selective encryption for enforcing selective access privileges, which supports both read and write operations. To effectively support updates to the access control policy, while limiting expensive re-encryption operations, this chapter also illustrates over-encryption as a solution to partially delegate to the cloud provider grant and revoke operations.
- Chapter 3 specifically focuses on the supply chain scenario, illustrating how RFID tags can be used for authentication and how selective encryption can enforce selective access to information about objects by the players of a supply chain.
- Chapter 4 presents concluding remarks.

2. Selective access

In this chapter, we illustrate how a data owner outsourcing her resources to a Cloud Service Provider (CSP) can selectively make them accessible to other users by means of selective encryption, which adopts different encryption keys for different resources, and distributes keys to users so that each user can decrypt only allowed resources. We first illustrate how selective encryption, coupled with suitable key derivation techniques, operates to enforce access control (Section 2.1). We then discuss how updates to the access control policy (Section 2.2) and write access restrictions (Section 2.3) are enforced. We finally discuss a possible approach for the implementation of the proposed access control system in OpenStack (Section 2.4).

2.1 Selective encryption for access control

Our solution for enforcing access control to outsourced data is based on *selective encryption*. Selective encryption consists in using different keys to encrypt different tuples, and in selectively distributing those keys to authorized users so that each user can decrypt (and therefore access) all and only the tuples she is entitled to access according to the authorization policy defined by the data owner. The information that a data owner can outsource to a CSP can be of any type: relational databases, XML documents, multimedia files, and so on. For simplicity, but without loss of generality, in this chapter we assume the data outsourced to the CSP to be organized in a relational database, with the note that the approach illustrated in the following can be easily adapted to operate on any logical data modeling. We then consider a relation r defined over schema $R(a_1,\ldots,a_n)$, where attribute a_i is defined over domain D_i, $i=1,\ldots,n$. At the instance level, each relation r is composed of a set of tuples, where each tuple t is a function mapping attributes to values in their domains. Given an attribute a, notations $t[a]$ represent the value of attribute a in t. At the CSP, relation r is represented through an encrypted relation r^k, defined over schema R^k(<u>`tid`</u>, `enc`), with `tid` a numerical primary key added to the encrypted relation and `enc` the encrypted tuple. Each tuple t in r is represented as an encrypted tuple t^k in r^k, where t^k[`tid`] is randomly chosen by the data owner and t^k[`enc`]=$E_k(t)$, with E a symmetric encryption function with key k.

Given a set U of users and a relation r to be outsourced, the authorization policy regulating which user $u \in U$ can read which tuple $t \in r$, is defined before outsourcing relation r (e.g., [DEF$^+$14, DFJL12, DFJ$^+$10]). The authorization policy can be represented as a binary *access matrix M* with a row for each user u, and a column for each tuple t, where: $M[u,t]$=1 iff u can access t; $M[u,t]$=0 otherwise. To illustrate, consider relation PATIENTS in Figure 2.1. Figure 2.2 illustrates an example of access matrix regulating access to the tuples in relation PATIENTS by users A, B, C, D, and E. The j-th column of the matrix represents the access control list $acl(t_j)$ of tuple t_j, for each $j=1,\ldots,|r|$. As an example, with reference to the matrix in Figure 2.2, $acl(t_1)$=ABC.

PATIENTS

	SSN	Name	ZIP	MarStatus	Illness
t_1	123456789	Ann	22010	single	gastritis
t_2	234567891	Barbara	24027	divorced	neuralgia
t_3	345678912	Carl	22010	married	gastritis
t_4	456789123	Daniel	20100	married	gastritis
t_5	567891234	Emma	21048	single	neuralgia
t_6	678912345	Fred	23013	married	hypertension
t_7	789123456	Gary	22010	widow	gastritis
t_8	891234567	Harry	24027	widow	hypertension

Figure 2.1: An example of relation

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
A	1	1	0	1	1	1	1	0
B	1	1	1	1	1	0	0	0
C	1	1	1	0	1	1	0	0
D	0	0	0	1	1	1	0	1
E	0	0	0	1	1	1	0	0

Figure 2.2: An example of access matrix

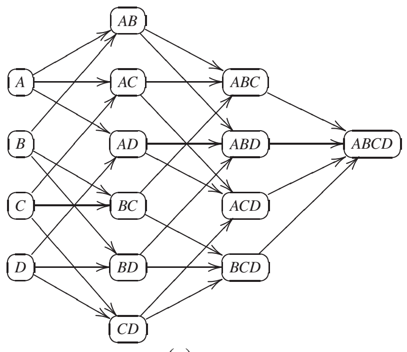
Enforcing an access control policy with encryption requires to define an *encryption policy*, to establish keys for encrypting tuples and keys to be distributed to users. Indeed, the encryption policy must be *equivalent* to the authorization policy, meaning that each user should be able to decrypt all and only the tuples she is authorized to access. Solutions translating an authorization policy into an equivalent encryption policy (e.g., [DFJ$^+$10]) have two main design desiderata: *i)* guarantee that each user has to manage only one key; and *ii)* encrypt each tuple with only one key (i.e., no tuple is replicated).

To fulfill these two requirements, selective encryption approaches rely on *key derivation techniques* that allows a user to compute an encryption key k_j starting from the knowledge of another key k_i and of a piece of publicly available information. To determine which key can be derived from which other key, key derivation techniques require the preliminary definition of a *key derivation hierarchy*. A key derivation hierarchy can be graphically represented as a directed acyclic graph with a vertex v_i for each key k_i in the system and an edge (v_i,v_j) from key k_i to key k_j iff k_j can be directly derived from k_i. Note that key derivation can be applied in chain, meaning that key k_j can be computed starting from key k_i if there is a path (of arbitrary length) from v_i to v_j in the key derivation hierarchy.

A key derivation hierarchy can have different shapes, as described in the following.

- *Chain of vertices* (e.g., [San87]): the key k_j associated with vertex v_j is computed by applying a one-way function to the key k_i of its predecessor in the chain (e.g., $k_j = h(k_i)$, with h a one-way hash function). With this kind of shape of the key derivation structure, no public information is needed.

- *Tree hierarchy* (e.g., [San88]): the key k_j associated with vertex v_j is computed by applying a one-way function to the key k_i of its direct ancestor, and a public label l_j associated with k_j (e.g., $k_j = h(k_i,l_j)$, with h a one-way hash function). Public labels are necessary to guarantee that different children of the same node in the tree have different keys.



(a)

user	key
A	k_A
B	k_B
C	k_C
D	k_D

(b)

tuple	key
t_1	k_{ABC}
t_2	k_{ABC}
t_3	k_{BC}
t_4	k_{ABD}
t_5	k_{ABCD}
t_6	k_{ACD}
t_7	k_A
t_8	k_D

(c)

Figure 2.3: An example of encryption policy equivalent to the access control policy in Figure 2.2, considering the subset $\{A,B,C,D\}$ of users

- *DAG hierarchy* (e.g., [AT83, ABFF09, CMW06, DFM04]): since a vertex v_j in a DAG can have more than one direct ancestor, to enable the derivation of key k_j from the keys of different direct ancestors, each edge in the hierarchy is associated with a publicly available *token* [ABFF09]. Given two keys k_i and k_j, and the public label l_j of k_j, $t_{i,j}$ permits to compute k_j from k_i and l_j, and is computed as $t_{i,j}{=}k_j{\oplus}h(k_i,l_j)$, where $\oplus$ is the bitwise XOR operator, and h is a deterministic cryptographic function. By means of $t_{i,j}$, all users knowing (or able to derive) key k_i can also derive key k_j.

Each of the proposed shapes (and hence, each of the corresponding key derivation techniques) has advantages and disadvantages. However, the DAG hierarchy, which adopts tokens for key derivation, best fits the outsourcing scenario by minimizing the need of re-encryption and/or key re-distribution in case of updates to the authorization policy [DFJ+10] (for more details, see Section 2.2).

An intuitive approach to define a DAG key derivation hierarchy suited for access control enforcement, and able to satisfy the desiderata of limiting the key management overhead, adopts the set containment relationship $\subseteq$ over the set U of users [DFJ+10]. Such a hierarchy has a vertex for each of the elements of the power-set of the set U of users, and a path from v_i to v_j iff the set of users represented by v_i is a subset of that represented by v_j. The correct enforcement of the authorization policy defined by the data owner is guaranteed iff: *i)* each user u_i is communicated the key associated with the vertex representing her; and *ii)* each tuple t_j is encrypted with the key of the vertex representing $acl(t_j)$. With this strategy, each tuple can be decrypted and accessed by all and only the users in its access control list, meaning that the encryption policy is equivalent to the authorization policy defined by the data owner. Furthermore, each user has to manage one key only, and each tuple is encrypted with one key only. For instance, Figure 2.3(a) illustrates the key derivation hierarchy induced by the set $U{=}\{A,B,C,D\}$ of users and the subset containment relationship over it (in the figure, vertices are labeled with the set of users they represent). Figure 2.3(b) and Figure 2.3(c) illustrate the keys assigned to users in the system and the keys used to encrypt the tuples in relation PATIENTS in Figure 2.1, respectively. The encryption policy in the figure enforces the access control policy in Figure 2.2 restricted to the set $U{=}\{A,B,C,D\}$ of users as each user can derive, from her own key, the keys of the vertices to which she belongs and hence decrypt the tuples she is authorized to read. For instance, user C can derive the keys used to encrypt tuples t_1, t_2, t_3, t_5, and t_6, and then access their content.

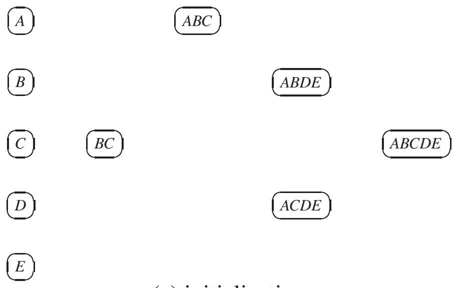
Even though this approach correctly enforces an authorization policy and enjoys ease of implementation, it defines more keys and more tokens than necessary. Since tokens are stored in a publicly available catalog at the CSP, when a user u wants to access a tuple t she needs to interact with the CSP to visit the path in the key derivation hierarchy from the vertex representing u to the vertex representing $acl(t)$. Therefore, keeping the number of tokens low increases the efficiency of the derivation process, and then reduces the response time to users. The problem of minimizing the number of tokens, while guaranteeing equivalence between the authorization and the encryption policies, is NP-hard (it can be reduced to the set cover problem) [DFJ$^+$10]. It is however interesting to note that not all the vertices and tokens in the key derivation hierarchy discussed above are necessary for the enforcement of an access control policy.

- The vertices needed for correctly enforcing an authorization policy are only those representing singleton sets of users (corresponding to users' keys) and the access control lists of the tuples (corresponding to keys used to encrypt tuples) in r. For instance, with respect to the encryption policy in Figure 2.3, vertex AB represents neither a user of the system nor the access control list of a resource (Figures 2.3(b)-(c)), and is therefore not necessary for the enforcement of the access control policy.
- It is sufficient that there exists a path reaching the vertex representing the access control lists of a tuple form the vertex of each of the users who can access the tuple. For instance, with respect to the encryption policy in Figure 2.3, one among edges (AB,ABD), (AD,ABD), and (BD,ABD) is redundant.
- When two or more vertices have more than two common direct ancestors, the insertion of a vertex representing the set of users corresponding to these ancestors reduces the total number of tokens.

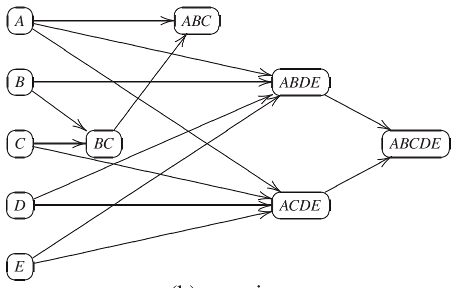
Elaborating on these two intuitions to reduce the number of tokens, the following heuristic approach efficiently provides good results [DFJ$^+$10].

1. *Initialization*. The algorithm first identifies the vertices necessary to enforce the authorization policy, that is, the vertices representing: *i)* singleton sets of users, whose keys are communicated to users and that allow them to derive the keys of the tuples they are entitled to access; and *ii)* the access control lists of the tuples, whose keys are used for encryption. These vertices represent the set of *material* vertices of the system.
2. *Covering*. For each material vertex v corresponding to a non-singleton set of users, the algorithm finds a set of material vertices that form a *non-redundant set covering* for v, which become direct ancestors of v. A set V of vertices is a set covering for v if for each u in v, there is at least a vertex v_i in V such that u appears in v_i. It is non-redundant if the removal of any vertex from V produces a set that does not cover v.
3. *Factorization*. For each set $\{v_1,\ldots v_m\}$ of vertices that have $n > 2$ common ancestors $v'_1,\ldots,v'_n$, the algorithm inserts an intermediate vertex v representing all the users in $v'_1,\ldots,v'_n$ and connects each v'_i, $i = 1,\ldots,n$, with v, and v with each v_j, $j = 1,\ldots,m$. In this way, the encryption policy includes $n+m$, instead of $n \cdot m$ tokens in the catalog.

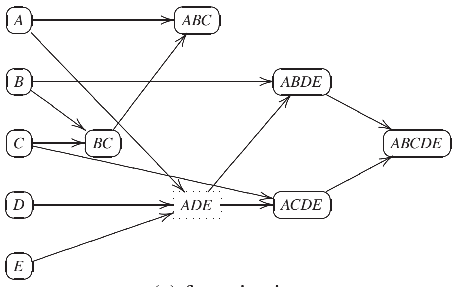
Figure 2.4 illustrates, step by step, the definition of the key derivation hierarchy through the algorithm in [DFJ$^+$10], for the authorization policy in Figure 2.2. The initialization phase generates



(a) initialization



(b) covering



(c) factorization

user	key
A	k_A
B	k_B
C	k_C
D	k_D
E	k_E

tuple	key
t_1	k_{ABC}
t_2	k_{ABC}
t_3	k_{BC}
t_4	k_{ABDE}
t_5	k_{ABCDE}
t_6	k_{ACDE}
t_7	k_A
t_8	k_D

(d) key assignment

Figure 2.4: Definition of an encryption policy equivalent to the access control policy in Figure 2.2

the set of (material) vertices in Figure 2.4(a). The covering phase generates the preliminary key derivation hierarchy in Figure 2.4(b), where each vertex is connected to a set of parents including all and only the users in the vertex itself. The factorization phase generates the key derivation hierarchy in Figure 2.4(c), which has an additional non-material vertex (i.e., ADE, denoted with a dotted line in the figure) representing the users that belong to both $ABDE$ and $ACDE$. This factorization saves one token. Figure 2.4(d) illustrates the keys assigned to users in the system and the keys used to encrypt the tuples in relation PATIENTS in Figure 2.1.

The key derivation hierarchy obtained through the algorithm illustrated above is stored at the CSP through a *token catalog* that keeps track, for each edge (v_i,v_j) in the hierarchy, of the label of the source l_i and destination l_j nodes and of the token value $token_{i,j}$. When a user u needs to access a tuple t, she need to retrieve a chain of tokens that, starting from her own key k_u, ends in the key used to encrypt t. To this purpose, user u interacts with the server. Given the label associated with key k_u and the tuple t that the user wants to access, the server first retrieves the label of the key k_t used to encrypt t. Then, it retrieves the shortest path from k_u to k_t in the key and token graph through a shortest path algorithm, by properly querying the token catalog. The server then returns to the user the encrypted tuple t^k and the tokens composing the shortest path. The user will use these tokens to derive key k_t starting from her key k_u, to finally decrypt t^k.

2.2 Policy updates

In case of changes to the authorization policy, the encryption policy must be updated accordingly, to guarantee their equivalence. This section illustrates how the selective encryption approach presented above can enforce grant and revoke operations in the authorization policy.

2.2.1 Over-encryption

Since the key used to encrypt each tuple t in r depends on the set of users who can access it, it might be necessary to re-encrypt the tuples involved in the policy update with a different key that only the users in their new access control lists know or can derive. A trivial approach to enforce a grant/revoke operation on tuple t requires the data owner to: *i)* download t^k from the CSP; *ii)* decrypt it; *iii)* update the key derivation hierarchy if it does not include a vertex representing the new set of users in $acl(t)$; *iv)* encrypt t with the key k' associated with the vertex representing $acl(t)$; *v)* upload the new encrypted version of t on the CSP; and *vi)* possibly update the public catalog containing the tokens. For instance, consider the encryption policy in Figures 2.4(c-d) and assume that user D is granted access to tuple t_1. The data owner should download t_1^k; decrypt it using key k_{ABC}; insert a vertex representing $acl(t_1)$=$ABCD$ in the key derivation hierarchy; encrypt t_1 with k_{ABCD}; and upload the encrypted tuple on the CSP, together with the tokens necessary to users A, B, C, and D to derive k_{ABCD}. This approach, while effective and correctly enforcing authorization updates, leaves to the data owner the burden of managing the update and requires a heavy interaction between the data owner and the CSP. Also, re-encryption operations are computationally expensive. To limit the data owner overhead, we propose to adopt two layers of encryption (each characterized by its own encryption policy), to partially delegate to the CSP the management of grant and revoke operations [DFJ+10].

- The *Base Encryption Layer* (BEL) is applied by the data owner before outsourcing the dataset. A BEL key derivation hierarchy is built according to the authorization policy existing at initialization time. In case of policy updates, BEL is only updated by possibly inserting tokens in the public catalog (i.e., edges in the BEL key derivation hierarchy). Note that each vertex v in the BEL key derivation hierarchy has two keys: a derivation key k (used for key derivation only), and an access key k^a (used to encrypt tuples, but that cannot be exploited for key derivation purposes).

- The *Surface Encryption Layer* (SEL) is applied by the CSP over the tuples that have already been encrypted by the data owner at BEL. It dynamically enforces the authorization policy updates by possibly re-encrypting tuples and changing the SEL key derivation hierarchy to correctly reflect the updates. Differently from BEL, vertices in the SEL key derivation hierarchy are associated with a single key k^s.

Intuitively, with the over-encryption approach, a user can access a tuple t only if she knows the keys used to encrypt t at BEL and SEL. Two basic approaches (i.e., Full_SEL and Delta_SEL) can be adopted in the construction of the two levels, as briefly described in the following.

- With the Full_SEL approach, 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 at BEL corresponds to a graph being isomorphic to the one existing at BEL. The SEL policy initially models exactly the BEL policy, and hence by definition is correct (at initialization time) with respect to the access control policy. When the existing authorization policy changes, the SEL policy is updated to reflect the grant/revoke operation. Hence, at any time, the BEL level enforces an encryption policy that is equivalent to the authorization policy.

- With the Delta_SEL approach, the SEL policy is initialized to not carry out any over-encryption. Hence, at initialization time the SEL level does not add any protection, but enforces a double encryption only when a change in the existing authorization policies of a data owner is requested.

Depending on whether the Full_SEL or Delta_SEL approach is employed, the over-encryption operation is performed in two different ways. In the Full_SEL case, the SEL encryption layer is employed to deny the access to a tuple t to all the users but the ones in $acl(t)$. This is done regardless of the users being allowed or not to access t, through being able to derive the related BEL key. By contrast, in the Delta_SEL approach, the locking effect of the SEL layer is employed only in order to limit the access rights granted by the BEL keys owned by no longer authorized users.

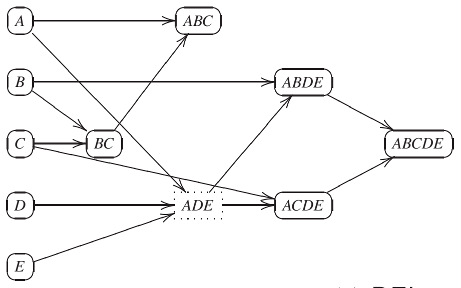
Both the Full_SEL and the Delta_SEL approaches produce a correct two layer encryption: given a correct encryption policy at the BEL level, both the approaches produce a SEL level such that the encryption policies at the two levels (jointly) enforce the authorizations. The reason for considering both the approaches lies in the different performance and protection guarantees 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 2.2.3, the Full_SEL approach provides more protection as it is not affected by a risk of information exposure characterizing the Delta_SEL approach. The choice between one or the other can then be a trade-off between costs and resilience to attacks.

We note that, besides the Full_SEL and Delta_SEL approaches, a third strategy 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 CSP's eyes. However, this latter approach is of no interest as it presents a significant exposure to collusion (see Section 2.2.3).

2.2.2 Grant and revoke

Adopting the two-layer encryption approach illustrated above, grant and revoke operations over an authorization policy are enforced as follows.

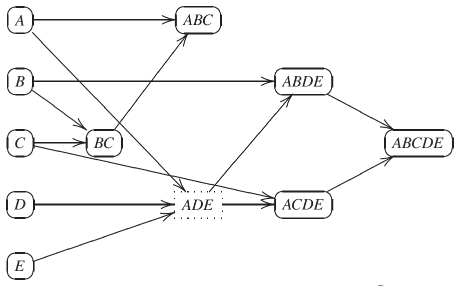
- *Grant*. When user u is granted access to tuple t, she needs to know the key used to encrypt t at both BEL and SEL. Hence, the data owner adds a token in the BEL key derivation hierarchy from the key of user u (i.e., key of the vertex representing u) to the access key used to encrypt t (i.e., the vertex representing $acl(t)$ at initialization time). The owner then asks the CSP to update the key derivation hierarchy at SEL and to possibly re-encrypt tuples. If we adopt a Full_SEL approach, tuple t is encrypted at SEL with the key of the vertex representing $acl(t) \cup \{u\}$ (which is possibly inserted into the hierarchy). Besides t, also other tuples may need to be re-encrypted at SEL to guarantee the correct enforcement of the policy update. In fact, tuples that are encrypted with the same key as t at BEL and that user u is not allowed to read must be encrypted at SEL with a key that u does not know (and cannot derive). The data owner must then make sure that each tuple t_i sharing the BEL encryption key with t are encrypted at SEL with the key of the vertex representing $acl(t_i)$. If, on the contrary, we adopt a Delta_SEL approach, only tuple t does not need to be encrypted at SEL, since BEL already provides access only to authorized users. On the contrary, each



user	key
A	k_A
B	k_B
C	k_C
D	k_D
E	k_E

tuple	key
t_1	k^a_{ABC}
t_2	k^a_{ABC}
t_3	k^a_{BC}
t_4	k^a_{ABDE}
t_5	k^a_{ABCDE}
t_6	k^a_{ACDE}
t_7	k^a_{A}
t_8	k^a_{D}

(a) BEL



user	key
A	k^s_A
B	k^s_B
C	k^s_C
D	k^s_D
E	k^s_E

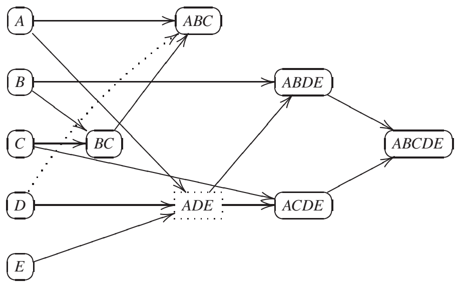
tuple	key
t_1	k^s_{ABC}
t_2	k^s_{ABC}
t_3	k^s_{BC}
t_4	k^s_{ABDE}
t_5	k^s_{ABCDE}
t_6	k^s_{ACDE}
t_7	k^s_{A}
t_8	k^s_{D}

(a) SEL

Figure 2.5: Encryption policies at BEL and SEL, equivalent to the access control policy in Figure 2.2

tuple t_i sharing the BEL encryption key with t must be encrypted at SEL with the key of the vertex representing $acl(t_i)$. For instance, consider the access matrix in Figure 2.2 and the encryption policies at BEL and SEL enforcing it in Figure 2.5, and assume that user D is granted access to tuple t_1. Figure 2.6 illustrates the encryption policies at BEL and SEL after the enforcement of the grant operation, assuming to apply a Full_SEL strategy. To enforce this change in the access control policy, the data owner must first add a token that permits user D to derive the access key of vertex ABC (k^a_{ABC}) used to encrypt t_1 at BEL (dotted edge in the figure). Also, she will ask the CSP to update the SEL key derivation hierarchy to add a vertex representing $ABCD$. Tuple t_1 is then over-encrypted at SEL with the key of this new vertex.

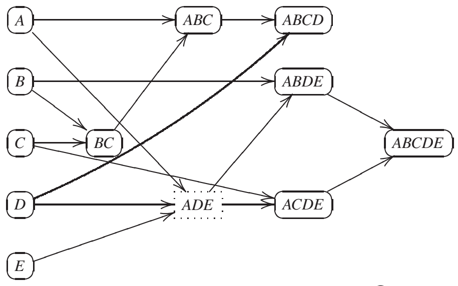
- *Revoke.* In both Full_SEL and Delta_SEL, when user u loses the privilege of accessing tuple t, the data owner simply asks the CSP to re-encrypt (at SEL) the tuple with the key associated with the set $acl(t) \setminus \{u\}$ of users. If the vertex representing this set of users is not represented in the SEL key derivation hierarchy, the CSP first updates the hierarchy inserting the new vertex, and then re-encrypts the tuple. For instance, consider the encryption policies at BEL and SEL in Figure 2.6 and assume that the data owner revokes B the privilege to access t_4. The data owner requires the CSP to change SEL (BEL is not affected by revoke operations) to guarantee that tuple t_4 is encrypted with a key that user B cannot derive. To this aim, t_4 is re-encrypted with key k^s_{ADE}. Figure 2.7 illustrates the encryption policies at BEL and SEL after the enforcement of the revoke operation, assuming to apply a Full_SEL approach. Note that vertex $ABDE$ is removed from the hierarchy since it is neither necessary for policy enforcement nor useful for reducing the number of tokens.



user	key
A	k_A
B	k_B
C	k_C
D	k_D
E	k_E

tuple	key
t_1	k^a_{ABC}
t_2	k^a_{ABC}
t_3	k^a_{BC}
t_4	k^a_{ABDE}
t_5	k^a_{ABCDE}
t_6	k^a_{ACDE}
t_7	k^a_A
t_8	k^a_D

(a) BEL

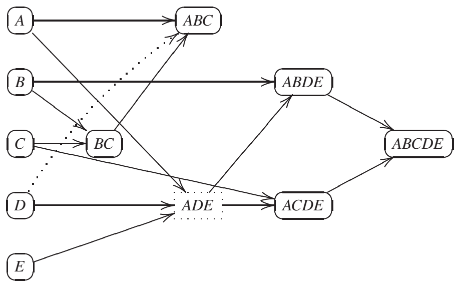


user	key
A	k^s_A
B	k^s_B
C	k^s_C
D	k^s_D
E	k^s_E

tuple	key
t_1	k^s_{ABCD}
t_2	k^s_{ABC}
t_3	k^s_{BC}
t_4	k^s_{ABDE}
t_5	k^s_{ABCDE}
t_6	k^s_{ACDE}
t_7	k^s_A
t_8	k^s_D

(a) SEL

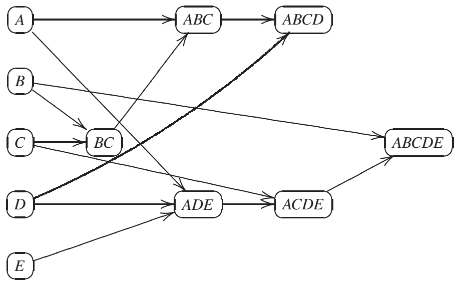
Figure 2.6: Encryption policies at BEL and SEL in Figure 2.5 after granting D access to t_1



user	key
A	k_A
B	k_B
C	k_C
D	k_D
E	k_E

tuple	key
t_1	k^a_{ABC}
t_2	k^a_{ABC}
t_3	k^a_{BC}
t_4	k^a_{ABDE}
t_5	k^a_{ABCDE}
t_6	k^a_{ACDE}
t_7	k^a_A
t_8	k^a_D

(a) BEL



user	key
A	k^s_A
B	k^s_B
C	k^s_C
D	k^s_D
E	k^s_E

tuple	key
t_1	k^s_{ABCD}
t_2	k^s_{ABC}
t_3	k^s_{BC}
t_4	k^s_{ADE}
t_5	k^s_{ABCDE}
t_6	k^s_{ACDE}
t_7	k^s_A
t_8	k^s_D

(a) SEL

Figure 2.7: Encryption policies at BEL and SEL in Figure 2.6 after revoking B access to t_4

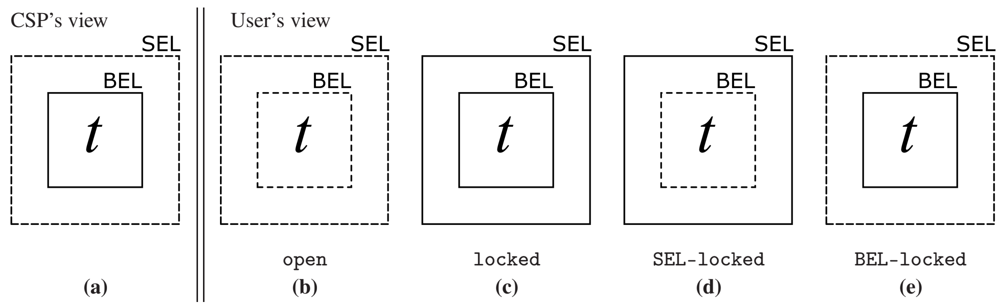


Figure 2.8: Possible views on a tuple t

2.2.3 Exposure evaluation

Since the management of (re-)encryption operations at SEL is delegated to the CSP, there is the risk of collusions with users. In fact, by combining their knowledge, a user and the CSP can possibly decrypt tuples that neither the CSP nor the user can access. For instance, with reference to the encryption policy in Figure 2.7, the CSP and user D can access to tuple t_2 by combining their knowledge. In fact, this tuple is encrypted with access key k^a_{ABC} at BEL, known to user D as it is used to encrypt t_1, and with key k^s_{ABC} at SEL, known to the CSP. Collusion represents a risk to the correct enforcement of the authorization policy but, as will be clarified in the following, such risk is limited, well-identifiable, and possible to mitigate.

In order to model the information leakage due to collusions, we assume that users are not oblivious (i.e., they have the ability to store and keep indefinitely all information they were entitled to access). To examine the different views that each user can have on a tuple t, we employ a graphical notation with tuple t in the center and with fences around t denoting the barriers to the access imposed by the knowledge of the keys used for t's encryption at both the BEL level (inner fence) and the SEL level (outer fence). The fence is continuous if there is no knowledge of the corresponding key and it is discontinuous otherwise.

Figure 2.8(a) shows the view of the CSP itself, which knows the SEL-level key, but does not have access to the BEL-level key. Figures 2.8(b-e) represent the views of the users: the `open` view corresponds to the view of authorized users, while the remaining ones (`locked`, `SEL-locked`, `BEL-locked`) show the views of non-authorized users.

Collusion can take place every time two entities, combining their knowledge (i.e., the keys known to them) can acquire knowledge on the content of a tuple that *neither* of them is authorized to access. Therefore, users having the `open` view need not be considered as they have nothing to gain in colluding (they already access t). Following the same line of reasoning, users having the `locked` view will not be considered, since they have nothing to offer. The only possible scenario where collusion can happen is when two parties (the CSP and a user, or two users) having the `BEL-locked` and `SEL-locked` view over a tuple t share their knowledge of the SEL and BEL key used to protect t to gain access to a tuple that none of them is authorized to view. In the Full_SEL approach, no one but the CSP can have a `BEL-locked` view, while only users can have a `SEL-locked` view. In the Delta_SEL approach instead each user has the `BEL-locked` view over the tuples she is not authorized to access at initialization time.

There are only two reasons for which a user (in the Full_SEL or in the Delta_SEL scenario) can have the `SEL-locked` view on a tuple, as described in the following.

- *Revoke*. The user was previously authorized to access the tuple and the authorization was then revoked. In this case, since the user is supposed to be non oblivious, she has no gain in colluding with the CSP. It is therefore legitimate to consider this case ineffective with respect to collusion risks.[1]

- *Policy split*. The user has been granted the authorization for tuple t' that was, at initialization time, encrypted with the same key as t (i.e., $acl(t') \subseteq acl(t)$), leaving t `SEL-locked` [DFJ+07, DFJ+10]. In this situation, the user has never had access to t and must not be able to gain it, therefore there is indeed exposure to collusion.

It is interesting to note that the Full_SEL approach is exposed only to collusion between the CSP, holding the `BEL-locked` view over a t, and a user who acquired the `SEL-locked` view over t as a consequence of a policy split. The Delta_SEL approach is instead also exposed to the misbehavior of a single curious (planning-ahead) user. In fact, every user initially has the `BEL-locked` view over the tuples she is not authorized to access. This view can then evolve in the `open` view if she is granted access to t, or in the `SEL-locked` view she is granted access to t' that is encrypted with the same key as t at initialization time. Therefore, although more efficient as it limits the adoption of double encryption, Delta_SEL is more exposed to risks than Full_SEL approach. It is however 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 (i.e. organizing resources in such a way to minimize policy splits).

2.3 Support for write privileges

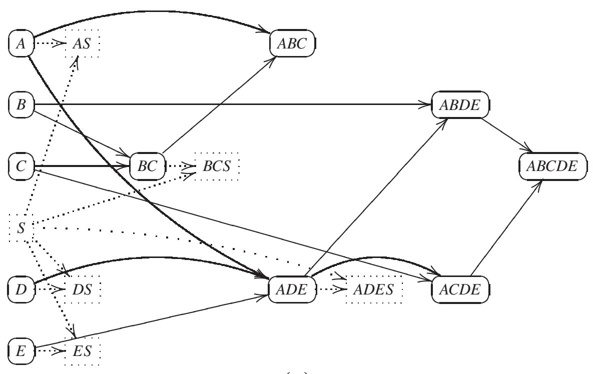
The solution described in the previous section, while effectively enforcing read privileges and updates to them, assumes the outsourced relation to be read-only (i.e., only the owner can modify tuples). To allow the data owner to selectively authorize other users to update the outsourced data, this approach has been complemented with a specific technique to manage write privileges.

2.3.1 Write tags

A straightforward solution for enforcing write authorizations might consist in simply outsourcing to the external server the authorization policy (for write privileges) as is. The server would then perform traditional (authorization-based) access control. This solution would however not be in line with the goal of outsourcing, aimed at minimizing the server's involvement and responsibility in access control enforcement. We then propose to exploit selective encryption for the enforcement also of write authorizations, with the cooperation of the CSP.

Our solution associates each tuple with a *write tag* (i.e., a random value independent from the tuple content) defined by the data owner [DFJ+13]. Access to write tags is regulated through selective encryption: the write tag of tuple t is encrypted with a key known only to the users authorized to write t (i.e., the users specified within its write access list, denoted $acl_w(t)$) and by the CSP. In this way, only the CSP and authorized writers have access to the plaintext value of the write tag of each tuple. The CSP will then accept a write request on a tuple when the requesting user proves knowledge of the corresponding write tag.

[1]We assume, without loss of generality, that any time a tuple is updated, the data owner encrypts it with another BEL key as if it were a new tuple.



user	key
A	k_A
B	k_B
C	k_C
D	k_D
E	k_E
S	k_S

tuple	read_key	write_key
t_1	k_{ABC}	k_{AS}
t_2	k_{ABC}	k_{BCS}
t_3	k_{BC}	k_{BCS}
t_4	k_{ABDE}	k_{ADES}
t_5	k_{ABCDE}	k_{DS}
t_6	k_{ACDE}	k_{ES}
t_7	k_A	k_{AS}
t_8	k_D	k_{DS}

(a) (b) (c)

Figure 2.9: Encryption policy in Figures 2.4(c-d) extended to the enforcement of write privileges

Since the key used for encrypting the write tag of a tuple has to be shared between the CSP and the tuple writers, it is necessary to extend the key derivation hierarchy with the storing CSP. However, the CSP cannot access the outsourced tuples in plaintext, and hence it cannot be treated as an additional authorized user (i.e., with the ability of deriving keys in the hierarchy). The keys used to encrypt write tags are then defined in such a way that: *i)* authorized users can compute them applying a secure hash function to a key they already know (or can derive via a sequence of tokens); and *ii)* the CSP can directly derive them from a key k_S assigned to it, through a token specifically added to the key derivation hierarchy. Note that keys used for encrypting write tags cannot be used to derive other keys in the hierarchy. For instance, consider the encryption policy in Figures 2.4(c-d) and assume that $acl_w(t_1)$=$acl_w(t_7)$=A, $acl_w(t_2)$=$acl_w(t_3)$=BC, $acl_w(t_4)$=ADE, $acl_w(t_5)$=$acl_w(t_8)$=D, and $acl_w(t_6)$=E. Figure 2.9(a) illustrates the key derivation hierarchy, extended with the key k_S assigned to the CSP S and the keys necessary to encrypt write tags (the additional vertices and edges are dotted in the figure). Figures 2.9(b-c) summarize the keys assigned to users and to the CSP, and the keys used to encrypt the tuples in relation PATIENTS and their write tags, respectively.

2.3.2 Write policy updates

The over-encryption approach (Section 2.2.2), while effective for enforcing updates to a read authorization policy, is not suited for the enforcement grant and revoke of write authorizations. In fact, grant and revoke of write privileges need to operate on write tags, which are known to both authorized writers and to the CSP. Alternative solutions are therefore more effective and efficient. The solution we propose to enforce changes in write authorization operates as follows [DFJ+13].

- *Grant.* When user u is granted the privilege to modify tuple t, the write tag of t is encrypted with a key known to the CSP and the users in $acl_w(t) \cup \{u\}$. If the key derivation hierarchy does not include it, such a key is created and properly added to the hierarchy. For instance, with reference to the encryption policy in Figure 2.9, assume that user B is granted the write privilege over t_4. The write tag of the tuple needs to be encrypted with key k_{ABDES}, which is inserted into the key derivation hierarchy, while key k_{ADES} can be removed.

- *Revoke.* When user u is revoked the write privilege over tuple t, a fresh write tag must be defined for t, having a value independent from the former tag (e.g., it can be chosen adopting a secure random function). This is necessary to ensure that u, who is not oblivious, cannot exploit her knowledge of the former write tag of tuple t to perform unauthorized write operations. After the tag has been generated, it is encrypted with a key known to the CSP and to the users in $acl_w(t)\setminus\{u\}$. For instance, with reference to the encryption policy in Figure 2.9, assume that user C is revoked the write privilege over t_3. The write tag of the tuple needs to be changed and encrypted with key k_{BS}, which should be inserted into the key derivation hierarchy.

Note that, since the CSP is authorized to know the write tag of each and every tuple to correctly enforce write privileges, the data owner can delegate to the storing CSP both the generation and encryption (with the correct key) of the write tag of the tuples [DFJ+13]. Even if the CSP is assumed to be trustworthy in the management of write operations, it is important to provide a means to the data owner to verify that the CSP and users behaved correctly. In fact, the CSP could permit (for laziness or maliciously) malicious users to modify resources that they are not authorized to write. It is however possible to enable the data owner, authorized writers, and authorized readers to detect unauthorized modifications by complementing each tuple with proper integrity verification tags computed using HMAC functions (e.g., [DFJ+13]).

2.4 Implementation of the proposed approach

The approach discussed above will be the basis for the realization in OpenStack of a transparent service for the encryption of objects with encryption keys that are only accessible to the users who appear in the access control list associated with the object. The work uses as a foundation the implementation of the tools for the protection of data at rest developed within WP2 and extends them to the consideration of the access control policy and its evolution.

OpenStack is an open source project that aims at becoming an *operating system for the cloud*. It integrates a large collection of components, offering support to many tasks that are executed by the managers of a cloud infrastructure. For instance, components exist that support the activation of virtual machines (*Nova*), monitor their load and responsiveness (*Ceilometer*), help configure the network (*Neutron*), offer Web interfaces to the configuration of services (*Horizon*). The components that are of major interest for the realization of an encryption service consistent with the access policy are *Swift*, *Keystone*, and *Barbican*.

- *Swift* provides object storage to every component of the architecture and to applications built using OpenStack as a target. It uses clusters of standardized servers capable of storing petabytes of data. It uses a high level of redundancy in order to provide high availability. The access to objects occurs through a REST API.

- *Keystone* offers identity management services for the framework. It provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system and can integrate with existing backend directory services like LDAP. It supports multiple forms of authentication including standard username and password credentials and token-based systems.

- *Barbican* offers a key management service, usable by components internal to the OpenStack system and by external applications. It manages the secure storage and provisioning of secrets, such as passwords, encryption keys and X.509 Certificates. Consistently with the approach used by many components in OpenStack, it offers a REST API.

ESCUDO-CLOUD Use Case 1 aims at extending the support that OpenStack provides to the management of security requirements. The realization of the policy-based encryption services is one of the contributions that ESCUDO-CLOUD aims at producing in this domain.

The preliminary design of the implementation of the policy-based access control services for OpenStack is described in Work Document W2.2. This activity indeed spans both WP2 and WP3, because we combine an encryption-based protection of outsourced data (investigated by Task T2.1) with the support for the specification and evolution of the access control policy (investigated by Task 3.1).

It is to note that currently Swift only offers the support for access control at a relatively coarse level. Resources in Swift are organized into *Accounts*, *Containers* (within accounts), and *Objects* (within containers). Access control lists can be specified at the level of accounts and containers, and objects inherit the *acl* of the containers they are in. The *acl*s are represented as attributes of the containers, accessible using methods that return the properties of the container, in a JSON format. The support for encryption-based access control introduces additional *meta-containers*, one per every user, which are responsible to store the tokens that permit the derivation of the encryption keys. An important goal of the design of the system is to make the service *transparent* to current applications. The tokens are put into the meta-containers and read from them depending on the values of the *acl*s associated with the containers.

To support the insertion into the meta-containers of the tokens, a service is added to the OpenStack services. *RabbitMQ*, which is a messaging solution that offers persistence, is used to offer guarantees that the keys used to encrypt a resource are actually available to users interested in reading it. The persistence offered by RabbitMQ provides a guarantee of *eventual consistency*, which is consistent with the overall design of Swift.

More details on the design and implementation of these solutions can be found in Work Document W2.2. The work document initially provides a synthetic description of OpenStack and then describes the design of the software tools implementing the protection of data at rest in Swift. Work Document W2.2 presents only some initial considerations about the implementation of the container-level access control services based on encryption. A more extensive design will appear in Deliverable D2.2, which is due at M18.

3. Selective access for supply chain management

Following the supply chain nature of Use Case 2, we will apply the approach introduced in Section 2.1 for selective access to a supply chain scenario. For this, we will first outline a parts exchange scenario in Section 3.1. Afterwards, we will introduce *i)* how RFID tags can be utilized for authentication, and *ii)* how selective access is applied for ensuring authorization among multiple supply chain partners who are sharing data on multiple supply chain items in a single cloud architecture. We are confident that the combination of RFID authentication and selective access has the potential to overcome the challenges of integrity and security between supply chain collaborators who are utilizing a cloud architecture, thereby enabling new business opportunities through selective data sharing.

3.1 Problem statement

Imagine a set of mobile physical objects $O_1, \ldots, O_m$ each traversing a (potentially different) subset of players $X_1, \ldots, X_n$. Each player X_i collects information about each object O_j it handles (e.g., time, place, type of action, etc.) and stores this information in a central cloud database. Later other players may ask to access an object's data in this database. This scenario is common place in modern supply chains [SBE01]. Radio frequency identification (RFID) technology [Fin03] provides the means to equip and capture each object with an unique identifier. Data commonly collected typically includes time, location and type of handling (e.g., packing, unpacking, receiving, or shipping). On the one hand, combining these data from many companies (just predecessor and successor is almost always insufficient) along the supply chain enables or improves many economically attractive collaborative applications, such as batch recalls [WK08], counterfeit detection [STF05], benchmarking and analytics [Ker08, KDSB09, KOW10] or estimated arrival forecasts [CE09]. On the other hand, too liberal sharing of this information allows espionage on one's business operations [KTF08, DS08]. We observe this as a major obstacle to wider adoption of data sharing. Companies are usually part of many supply chains (even for the same product) all managed using a central cloud database. Specifying access control rules for this database can be very delicate. Consider the following two examples. Imagine a supplier S_1 selling a product p_1 to buyers B_1 and B_2. If B_1 has access to all scheduled orders for p_1, she can infer the volume of future business with B_2. This can be very sensitive, in case S_1 has to cancel some orders due to a temporary capacity reduction (e.g., a machine failure). B_1 could then infer whether B_2's orders are treated preferentially. While this decision can be based on local information in the case of bridging only one supply chain stage, it becomes difficult in case of a tier-2 supplier. Imagine a supplier S_2 selling product p_2 to S_1 which is then used to produce p_1. If either B_1 or B_2 contacts S_2 requesting data, S_2 cannot decide which object was shipped to which buyer. If S_2 would grant access to all items, B_1 could infer again the volume of business of B_2. A naturally emerging access rule is to share data with partners about shared objects, that is, objects both partners have possessed. This implements the important business concept of visibility, that is, each partner gains (additional) information about how its (entire) supplies are produced and how its (entire) products are used, but still provides a separation between different supply chains merging at one company. Furthermore it can be easily adopted reciprocally, i.e. “I give you access, if you give me access”, providing a fair allocation of cost, risk and benefits. We distinguish between downstream and upstream visibility. In downstream visibility a company is allowed to access data associated with its objects shipped to its supply chain partners (at those partners). Upstream visibility is the reverse and a company is allowed to access data associated with objects it has received from its supply chain partners (again at those partners). Setting these visibility policies correctly using existing access control models can be excruciatingly difficult, since the access control matrix can be huge $(n \times m)$ and each object can have a different trajectory (Section 3.2.2). First, we show how these policies can be implemented using a novel attribute in the framework of attribute-based access control (Section 3.2.3). Then we show how to implement this novel protocol with standard passive RFID tags (Section 3.3.1) and secure the data in a central cloud architecture by selective access (Section 3.3.2).

3.2 Supply chain visibility policies

3.2.1 Definition

We first formally define the trajectory of an object O_j and then define upstream and downstream visibility policies. Let there be n players $X_i \in \mathbb{X} = \{X_1, ..., X_n\}$. We model the trajectory $L(O_j) = \langle \mathbb{L}_j, \mathbb{R}_j \rangle$ of object O_j as a totally ordered set consisting of elements in the set $\mathbb{L}_j \subseteq \mathbb{X}$ and a binary relation $\mathbb{R}_j \subseteq \mathbb{L}_j \times \mathbb{L}_j$. The players represent the spatial domain of the trajectory and the players in $\mathbb{L}_j$ are those that have handled (possessed) the object O_j. The relation $\mathbb{R}_j$ models the temporal domain of the trajectory. Simply speaking, a player X_i is ranked lower than a player $X_{i'}$ in $L(O_j)$, that is, $\langle X_i, X_{i'} \rangle \in \mathbb{R}_j$, if X_i handled the object O_j earlier than $X_{i'}$. We write $\sigma(X_i, L(O_j))$ for a predicate that can be used to compute the rank of player X_i in $L(O_j)$ and $|\sigma(X_i, L(O_j))|$ for the evaluated rank itself. Then, $|\sigma(X_i, L(O_j))| < |\sigma(X_{i'}, L(O_j))|$ iff $\langle X_i, X_{i'} \rangle \in \mathbb{R}_j$. If player X_i did not handle the object O_j, then $|\sigma(X_i, L(O_j))|$ is undefined and any order relation on the natural numbers, e.g. both < and >, should always evaluate to false. We say the least element in $L(O_j)$ is the source of object O_j and the top most element is its destination. We can now capture the notion of being part of multiple supply chains that we briefly informally introduced in the introduction. A player X_i is part of multiple supply chains, if at least two objects that it handled have been handled by at least one player each - both upstream or downstream - which did not handle both objects. Formally iff X_i is part of two supply chains, then there exist two other players $X_{i'}$ and $X_{i''}$ and two objects O_j and $O_{j'}$, such that:

$$\begin{gathered} X_i \in \mathbb{L}_j, X_i \in \mathbb{L}_{j'} \\ X_{i'} \in \mathbb{L}_j, X_{i'} \notin \mathbb{L}_{j'} \\ X_{i''} \notin \mathbb{L}_j, X_{i''} \in \mathbb{L}_{j'} \end{gathered}$$

$$\begin{gathered} (|\sigma(X_i, L(O_j))| < |\sigma(X_{i'}, L(O_j))| \wedge |\sigma(X_i, L(O_{j'}))| < |\sigma(X_{i''}, L(O_{j'}))|) \vee \\ (|\sigma(X_{i'}, L(O_j))| < |\sigma(X_i, L(O_j))| \wedge |\sigma(X_{i''}, L(O_{j'}))| < |\sigma(X_i, L(O_{j'}))|) \end{gathered}$$

For simplicity we omit modeling a player handling an object more than once (e.g., product returns), although visibility policies are still valid and applicable. Now a player X_i is requesting information from player X_v (verifier) about object O_j stored in the cloud database. Player X_v intercepts this request and performs an access control decision. The intercepting component is called a policy enforcement point (PEP) and the information about the request, e.g. the identity of the requestor X_i and the unique identifier of the object O_j are forwarded to the policy decision point (PDP). The PDP compares the information to the policies in its store and returns its evaluation decision (grant or deny) to the PEP. As will be detailed in Section 3.3.2 the PEP will enforce a grant decision by re-encrypting its data on the object O_j within the cloud database.

Definition 1 *An* upstream visibility policy *grants (or denies) access to X_i for O_j based on the predicate evaluation*

$$|\sigma(X_i, L(O_j))| < |\sigma(X_v, L(O_j))|.$$

Definition 2 *A* downstream visibility policy *grants (or denies) access to X_i for O_j based on the predicate evaluation*

$$|\sigma(X_v, L(O_j))| < |\sigma(X_i, L(O_j))|.$$

3.2.2 Management of visibility policies

Existing access control models are faced with a serious scalability problem when protecting object-level data with visibility policies. There is a huge number of items (m) and each item requires unique protection corresponding to its trajectory. Its authorization matrix

$$player \times object \times access$$

is therefore huge and diverse at the same time, since there exists no natural grouping of objects. In order to simplify administration of visibility policies we can reduce the authorization matrix. Since the concept of visibility is independent of the players on the trajectory of an object, we can abstract from players in the authorization matrix and reduce it to

$$object \times visibility\ policy$$

Being even more restrictive we could further group objects, e.g., one can set a visibility policy for all objects of one product group. This reduces the authorization matrix to

$$object\text{-}group \times visibility\ policy$$

While this model is very restrictive, it is definitely also very simple and the administration effort is easily practically manageable in many real-world supply chains.

3.2.3 ABAC integration

Fortunately, we can combine this visibility policy authorization matrix with the expressiveness of sophisticated access control models without necessarily sacrificing simplicity of administration. The unifying model is attribute-based access control (ABAC) [NIS09]. This unification is important, since visibility policies can efficiently express access control policies necessary in many supply chains, but they are not sufficient in many cases. Companies may want to grant access to their object data to players outside of the supply chain. Examples are auditors or other service providers that provide outsourced services operating on object data. Also companies may want to restrict access to partners, although they are part of the supply chain. Examples are competitors or otherwise non-cooperating organizations. We therefore integrate the notion of visibility policies into the ABAC framework. Our integration is efficient and does not re-introduce the scalability problems described in Section 3.2.2. The ABAC framework integrates the features and expressive power of many access control models, including the classics RBAC, discretionary access control (DAC), and mandatory access control (MAC). We briefly review the policy model from Yuan and Tong [YT05] in a simplified form - we leave out environments and their attributes that are necessary to also include access control models such as TBAC or GTR-BAC.

- There are subjects and resources.
- $SA_l (1 \leq l \leq L)$ and $RA_m (1 \leq m \leq M)$ are the existing attributes for subjects and resources, respectively;
- $ATTR(s)$ and $ATTR(r)$ are attribute assignment relations for subject s and resource r, respectively:

$$ATTR(s) \subseteq SA_1 \times SA_2 \times \ldots \times SA_L$$
$$ATTR(r) \subseteq RA_1 \times RA_2 \times \ldots \times RA_M$$

- A policy rule that decides on whether a subject s can access a resource r, is a Boolean function of s's and r's attributes:

$$access(s,r) \leftarrow f(ATTR(s), ATTR(r))$$

  Given the attribute assignments of s and r, if the function f evaluates to true, then access to the cloud database resource entries is granted via re-encryption; otherwise access is denied as no re-encryption occurs.

Identity itself and roles become attributes of the subject and ownership in DAC becomes an attribute of the resource. Also, in MAC clearance becomes an attribute of the subject and classification an attribute of the resource. In our case subjects are the players X_i and the resources are objects O_j, but for mobile physical objects the situation is unfortunately slightly more complicated. The predicate $\sigma(X_i, L(O_j))$ in visibility policies depends on both player (X_i) and object (O_j). Therefore if we want to express upstream or downstream visibility as an attribute, we need to introduce a new type of attribute into the ABAC model. Our extension of the model is as follows:

- $SRA_k (1 \leq k \leq K)$ are the existing attributes for pairs of subjects and resources.
- $ATTR(s,r)$ is the attribute assignment relation for a pair of subject s and resource r:

$$ATTR(s,r) \subseteq SRA_1 \times SRA_2 \times \ldots \times SRA_K$$

- We extend the Boolean function for policy evaluation to s's, r's and both s and r's attributes:

$$access(s,r) \leftarrow f(ATTR(s), ATTR(r), ATTR(s,r))$$

We instantiate our ABAC model for visibility policies with the following two attributes: one for downstream and one for upstream partners.

$$SRA_h = \text{``}downstream\text{''} \qquad SRA_{h+1} = \text{``}upstream\text{''}$$

We can implement the assignment of these attributes as an evaluation of the predicates. When receiving $s = X_i, r = O_j, \sigma(X_v, L(O_j))$, and $\sigma(X_i, L(O_j))$ in an access request, player X_v can evaluate the predicates and assign corresponding attributes.

$$|\sigma(X_v, L(O_j))| < |\sigma(X_i, L(O_j)| \Rightarrow ATTR(s,r) = \text{“downstream”}$$
$$|\sigma(X_i, L(O_j))| < |\sigma(X_v, L(O_j)| \Rightarrow ATTR(s,r) = \text{“upstream”}$$

Our visibility policies can then be formulated as

$$access(s,r) \leftarrow \text{“downstream”} \in ATTR(s,r) \vee \text{“upstream”} \in ATTR(s,r)$$

We can also include or exclude supply chain partners as in our examples above

$$\begin{aligned} access(s,r) \leftarrow &\text{“auditor”} \in ATTR(s) \vee ((\text{“downstream”} \in ATTR(s,r) \vee \\ &\text{“upstreams”} \in ATTR(s,r)) \wedge \text{“competitor”} \notin ATTR(s)) \end{aligned}$$

3.3 RFID-authentication and selective encryption policy

We now introduce a two step solution to enforce the formulated attribute based access control. First, an authorization phase executed between a requestor and a data owner within a supply chain. Second, an authorization step performed by the data owner after successful authentication. While authentication is performed among supply chain members, authorization is performed strictly between the data owner and the central cloud repository to enforce selective access through re-encryption. After the authorization has been performed, the requestor can retrieve the demanded data from the central repository through a direct index lookup. As will be detailed in Section 3.3.1 the approach assumes that item level data sharing is performed through a single cloud architecture and that a trusted third party is incorporated into the authentication step.

3.3.1 Authentication via RFID

In order to enforce visibility policies the player must be able to reliably compute the predicate $\sigma(X_i, L(O_j))$ of an object's trajectory and supply it as an attribute in the access request to the PDP. This is challenging, since the necessary information about the trajectory is distributed across multiple players and need not be known to the player. Each player knows by default only her predecessor and successor from physically moving the object, which is even insufficient to determine its own rank in the trajectory. Therefore the requestor must supply the predicate, but of course the information is unreliable, since she should not necessarily be trusted. The player must verify the supplied predicate. This problem is an extension of the authentication problem in distributed systems. Clearly authentication is a prerequisite for any access control, but as seen before our predicate is an attribute of subject and resource, and as such common identity verification mechanisms fall short of solving the problem. Nevertheless, similar to the most common solutions for the authentication problem in distributed scenarios, we can resort to cryptographic techniques. We are concerned about physical objects (equipped with an RFID tag each) and a player needs to prove possession of this object. Differently from the ownership authentication factors - “something you have” - our authentication must succeed even if the player is no longer in possession of the object - “something you had”. This complicates the problem, since it rules out solutions of simply interactively using RFID for access control [Bly99, SWE03]. The notion of (current) possession has been explored before and extended to securely verifying ownership of an RFID tag [MSW05, Son08]. This concept already has many applications for mobile physical objects in supply chains, but, e.g. for any form of analytics, authentication and possession are decoupled. Also as pointed out in [vDMV09] these protocols still suffer from security flaws. The problem of authenticating based on (past) possession of RFID tags has been first considered in [KS09]. Yet these protocols do not allow implementing our predicate, but simply allow a decision on whether an item has been in possession. They therefore do not allow a distinction between upstream and downstream. Our authentication relies on a similar mechanism as the proofs of possession from [GA07]. A proof of possession is in our terms a verifiable predicate which can be supplied during the access request. Unfortunately all solutions proposed in [GA07] are either not reliable in our attacker model or are not realizable in our system model. A different design is therefore needed.

System Model

We continue our model with multiple players, but restrict ourselves in this section to one object O_j which is the one considered in the access request. One player (X_v) is the designated verifier of the predicate. We assume each player to be uniquely identifiable, and the availability of a public-key infrastructure to securely distribute public keys for each player. Besides the basic capabilities for communication, we only assume the availability of re-writable permanent storage on the RFID tag. Passive tags (without own source of power) with up to 64 KBytes of storage are available [Fuj08] and follow the EPC Gen 2 protocol which is commonly used in supply chains [EPC07]. Note that we do not consider cryptographic capabilities on the RFID tag, such as symmetric encryption [FWR05] or public-key cryptography [BGK$^+$07, BBD$^+$08, HWF08]. We emphasize that is a very strong restriction of the solution space. It implies that the RFID tag cannot manage secret material, such as cryptographic keys or even passwords. It therefore rules out any solution transferring common concepts from distributed systems authentication. For example, signing challenges by the RFID tag using message authentication codes or public-key signatures which significantly simplify the problem cannot be implemented in our model. We do this in order to address the security problems of existing and currently emerging deployments of RFID in supply chains which do not yet have these cryptographic capabilities. Before the access request and the problem of authentication, several operations are performed using the RFID tag. We use a simplified model from [KS09]. Assume Trent (T) is a trusted third party (TTP) that supports players in obtaining RFID tags. A natural choice is the RFID manufacturer. Note that Trent does not obtain any additional information about the supply chain operation than any RFID manufacturer already does now. Our authentication consists of the following algorithms or protocols.
Initialize: A player Alice requests a (or a set of) RFID tags from Trent. She can later use those to attach to newly created objects.
Move: A player Alice moves an object to another player Bob. She updates the information stored on the attached RFID tag. We emphasize that this operation does not require network access to Trent or Bob.
Authenticate: The requestor sends a verifiable predicate $\sigma(X_i, L(O_j))$ for access to the verifier. The verifier makes an access control decision based on this predicate (and its policies).

Security Model

We assume secure and authenticated communication over the network, that is, between the players and Trent. We assume insecure communication with the RFID tag attached to the object. Our attacker controls the requestor (X_i) and may control any other player except the verifier (X_v) and Trent (i.e., we consider almost arbitrary collusion). Our attacker is adaptive, that is, the set of controlled players may change over time.

Definition 3 *An admissible attacker A is a sequence of subsets $A_l \subset X(l = 1, ..., \lambda)$ of players none of which contains the verifier (X_v) or Trent:*

$$A_l \cap \{X_v, T\} = \emptyset.$$

Our main security guarantee is the non-forgeability of the verifiable predicate $\sigma(X_i, L(O_j))$. We state the following theorem.

Theorem 1 *No admissible attacker can forge a verifiable predicate $\sigma(X_i, L(O_j))$ for $X_i \in \cup_l A_l$ without possession of object O_j by any $X_{i'} \in \cup_l A_l$.*

Note that it is impossible to prevent forgery for an attacker in possession of object O_j. He can simply physical move the object to X_i. Even without physical movement, it is impossible to prevent an attacker from forgery, since she has access to all information (no trusted hardware) and can for instance, relay signals from the RFID tag. An interesting problem not captured by Theorem 1 arises when considering an adaptive adversary, due to the fact the attacker controls a fix set of parties. Assume that at time $\mathtt{t}_1$, X_i has possession of the object O_j and the attacker A has not compromised X_i, i.e. $X_i \notin A_{\mathtt{t}_1}$. Later at time $\mathtt{t}_2 > \mathtt{t}_1$ A compromises X_i, but now she should not be able to perform forgery of predicates for object O_j. We call this property non-transferability, since the attacker wants to transfer the predicate from X_i to another player $X_{i'}$. This property is similar to forward secrecy in key agreement protocols [DVW92]. Non-transferability is important, since it also prevents information leakages in the supply chain by otherwise trustworthy partners. For the formal definition we synchronize the time between the attacker and the trajectory. For each rank $l = 1, ..., |L_j|$ in the trajectory $L(O_j)$ there is a corresponding subset A_l of the attacker A. Since we cannot rely on cryptography on the RFID tag, we need to resort to different security measures. We cannot achieve cryptographic security for non-transferability in our model, because the attacker has access to all necessary information in the system at time $\mathtt{t}_1$ to produce a new system state for time $\mathtt{t}_2$ which cannot be disproved easily. Instead we rely on detection and traceability where more information is available. We refine Theorem 1 as follows.

Theorem 2 *Let Σ be the set of valid verifiable predicates $\sigma(X_i, L(O_j))$. No admissible attacker can prevent detection of $X_i \in \cup_l A_l, \sigma(X_i, L(O_j)) \in \Sigma$ after using a verifiable predicate $\sigma(X_{i'}, L(O_j)) \notin \Sigma$ as long as*

$$\forall \sigma(X_i, L(O_j)) \in \Sigma : X_i \notin A_{|\sigma(X_i, L(O_j))|}$$

Protocols

The protocols implementing the authentication are actually quite simple. We only use public-key signatures. Let $S_X()$ denote the signature with X's public key. Since we store the signatures on the RFID tag, it is beneficial to use very short signatures (e.g., [BLS04]). Our protocols chain the information along the object's trajectory. Each supplier is vouching for her buyer, similar to fourth factor authentication [BJR+06]. Fourth factor authentication relies on somebody you know in case other authentication factors have failed. Clearly a supplier knows her buyer and they trust each other at least to the extend to engage in business. We therefore use this fourth factor to replace the typical cryptographic authentication factor of knowledge - "something you know" - which is not available on the RFID tag.

Initialize: The trusted third party Trent T sends to player Alice an RFID tag with identifier j which contains in its storage the signature $S_T(j,A)$.

Move: A player Alice (A) wants to move an object to another player Bob (B). She appends to the storage on the RFID tag the recipient's identity (B) and her signature $S_A(j,B)$. Due to storage restrictions on RFID tags we recommend to compress the identity information as much as possible (e.g., using similar approaches as for abbreviation of URLs).

Authenticate: Let $s_1,\dots,s_k$ be the sequence of signatures stored on the RFID tag attached to object O_j. The requestor X_i sends as the verifiable predicate $\sigma(X_i,L(O_j))$ this sequence $s_1,\dots,s_k$. The verifier X_v verifies that

1. s_k conforms to $S_{X_{i_k}}(j,X_i)$.
2. For all $l(1<l<k)$ s_l conforms to $S_{X_{i_l}}(j,X_{i_{l+1}})$.
3. s_1 conforms to $S_T(j,X_{i_2})$.
4. For all $l(1\leq l\leq k)$ s_l is valid signature from X_{i_l}. If all checks are successful, then it evaluates its policies to make the access decision.

Figure 3.1 provides an example of supply chain operations based on the introduced authentication protocol. Assume a manufacturer M produces a good G. She first starts the **Initialize** protocol by contacting the trusted third party Trent T and requesting an RFID tag. T chooses a tag with identifier g, stores on the tag memory $s_1=S_T(g,M)$ and sends the RFID tag to M. M reads $\langle s_1\rangle$ from the tag, stores it in its database and attaches the tag to the good G. Now M intends to ship G to its distributor D. She starts the **Move** protocol and appends $s_2=S_M(g,D)$ to the tag's memory. M can then ship G to D. When D receives the good G, she reads $\langle s_1,s_2\rangle$ from the tag and stores it in her database. Later D may ship G to the retailer R. In this **Move** protocol she appends $s_3=S_D(g,R)$ to the tag's memory. R reads $\langle s_1,s_2,s_3\rangle$ from the tag of the received good G and stores it in his database. First consider the situation of a downstream request when M requests information from R, e.g., in order to collect sales data or analyze product returns. M and R run an **Authenticate** protocol. M reads $\langle s_1\rangle$ from her database and sends it along with g, M to R. R's PEP verifies the data in $\langle s_1\rangle$. It extracts M, g and the number of signatures (1) from $\langle s_1\rangle$. Then it looks up its corresponding predicate $\sigma(R,L(G))=\langle s_1,s_2,s_3\rangle$ in its database and similarly extracts R, g and the number of signatures (3).

3.3.2 Selective encryption within a supply chain scenario

In this section the selective encryption approach from Section 2.1 is applied to supply chain management. Consider again the simple supply chain illustrated in Figure 3.1, with T as TTP, M as manufacturer, D as distributor and R as retailer. Assume M produces a good G and moves it to D, who later moves it to R. As the good moves through the supply chain each player who possesses it collects information about it. We assume the players agreed on a common set of information (e.g., time, place, type of action etc.) that one can collect on an object, so that we can handle this information as a tuple. In a decentralized supply chain this information is stored in a local database. But in this case we want the information to be stored within a single cloud architecture

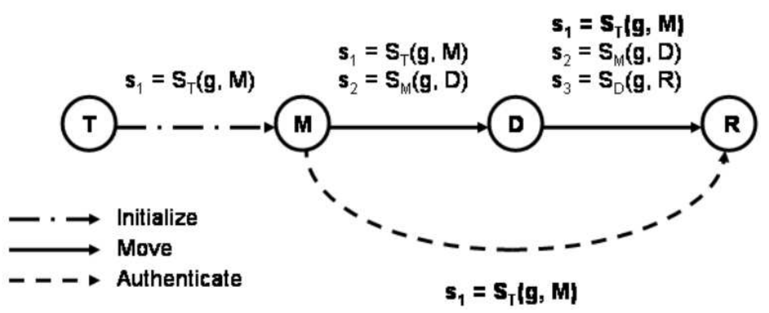


Figure 3.1: Example of an object traversing a supply chain

database. Due to the information being sensitive each player first chooses for each object that she handles a unique symmetric key k_O and encrypts the tuple before storing it in the database. Since the encryption policy requires that each user possesses only one encryption key, the player also chooses a unique private key k_U and generates a corresponding token according to the encryption policy. This allows key derivation of k_O from k_U. As the token are public information stored in the cloud database, the player retain only one key namely k_U.
To apply the selective encryption policy to the supply chain we follow the steps of the RFID-authentication protocol. We assume player $P \in \{M, D, R\}$ has a private key k_P. According to Section 2.1 we also assume that each tuple t_i has unique label l_i and the group G of users that has access to it has unique label l_G.

- **Initialize**: After receiving the RFID tag from T, M creates an object O. She chooses an encryption key k_1, collects information on O and stores the data as tuple t_1 encrypted with k_1 in the database. Then M creates $token_{M,1} = k_1 \oplus h(k_M, l_1)$ and stores it in the database.

- **Move**: M sends O to D. D forwards O to R.

  1. D generates a tuple t_2 and encrypts it with a newly chosen key k_2. Next, D creates $token_{D,2} = k_2 \oplus h(k_D, l_2)$.
  2. R generates a tuple t_3 and encrypts it with a newly chosen key k_3. Thereafter, R creates $token_{R,3} = k_3 \oplus h(k_R, l_3)$.

  Subsequent to initialize and move, the encryption policy of Section 2.1 is established. Due to the the tuple owner of t being the unique user who has access to t at this point of time the encryption policy is simple as illustrated in Figure 3.2.

- **Authentication**: Each player stores her information encrypted with a key that only she knows. Other players must authenticate to the tuple owner to be granted access to the information. Consider our example supply chain again and assume that M would like to read t_3. Then M has to authenticate to R using the RFID authentication protocol described in Section 3.3.1. If the authentication is successful, R updates the access control list of t_3, granting access to M. Now the group MR has access to t_3, therefore the tuple must be re-encrypted with a key k_{MR} that both M and R are able to derive with the corresponding token. However, since we are giving access to M there is no need to decrypt and re-encrypt t_3 with a new key. Hence in this case $k_{MR} = k_3$ holds and the token that R uses to derive it remains unchanged. The only token to be added is the token for M. Since the token computation contains private information, the two players have to collaborate to compute it. Player M

uses her private key k_M and the label of the group MR, which is public information, to compute $h(k_M, l_{MR})$. Afterwards she sends it securely to R. Finally R computes the token $k_{MR} \oplus h(k_M, l_{MR})$ and stores it in the public catalog.

$$\begin{array}{c|ccc|} & \textbf{t1} & \textbf{t2} & \textbf{t3} \\ \hline M & 1 & 0 & 0 \\ D & 0 & 1 & 0 \\ R & 0 & 0 & 1 \\ \hline \end{array}$$

(a) access matrix

user	key
M	k_M
D	k_D
R	k_R

(b) User Keys

tuple	key
t_1	k_1
t_2	k_2
t_3	k_3

(c) Tuple Keys

source	target	token
l_M	l_1	$k_1 \oplus h(k_M, l_1)$
l_D	l_2	$k_2 \oplus h(k_D, l_2)$
l_R	l_3	$k_3 \oplus h(k_R, l_3)$

(d) Tokens

Figure 3.2: Access matrix and encryption policy subsequently to the move step

$$\begin{array}{c|ccc|} & \textbf{t1} & \textbf{t2} & \textbf{t3} \\ \hline M & 1 & 0 & 1 \\ D & 0 & 1 & 0 \\ R & 0 & 0 & 1 \\ \hline \end{array}$$

user	key
M	k_M
D	k_D
R	k_R

tuple	key
t_1	k_1
t_2	k_2
t_3	k_{MR}

source	target	token
l_M	l_1	$k_1 \oplus h(k_M, l_1)$
l_M	l_{MR}	$k_{MR} \oplus h(k_M, l_{MR})$
l_D	l_2	$k_2 \oplus h(k_D, l_2)$
l_R	l_{MR}	$k_{MR} \oplus h(k_R, l_{MR})$

Figure 3.3: Access matrix and encryption policy after authentication of M for t_3

$$\begin{tikzcd} M \arrow[r] & MD \arrow[dr] & \\ D \arrow[ur] & & MDR \\ R \arrow[urr] & & \end{tikzcd}$$

$$\begin{array}{c|ccc|} & \textbf{t1} & \textbf{t2} & \textbf{t3} \\ \hline M & 1 & 1 & 1 \\ D & 1 & 1 & 0 \\ R & 1 & 0 & 0 \\ \hline \end{array}$$

source	target	token
l_M	l_{MD}	$k_{MD} \oplus h(k_M, l_{MD})$
l_M	l_3	$k_3 \oplus h(k_M, l_3)$
l_D	l_{MD}	$k_{MD} \oplus h(k_D, l_{MD})$
l_{MD}	l_{MDR}	$k_{MDR} \oplus h(k_{MD}, l_{MDR})$
l_R	l_{MDR}	$k_{MDR} \oplus h(k_R, l_{MDR})$

Figure 3.4: Key derivation graph and access matrix

Actually granting or revoking access to users, as illustrated in Section 2.2, may create new nodes or delete old ones that have become obsolete. Consider again our supply chain example depicted by Figure 3.2 and assume this time that the three tuples t_1, t_2, t_3 belong to M. Meaning, that three different objects left M and eventually reached D and R. Assume also that D and R were granted access to t_1, and only D can decrypt t_2. As illustrated in Figure 3.4, at a later time R also receives access to t_2. As there is already a node MDR the system is not required to add any token and just has to decrypt t_2 and encrypt it with k_{MDR}. However, now the node MD becomes obsolete. The node no longer represents the *acl* of a tuple and removing it reduces tokens as it has only one outgoing edge and more than one ingoing edge. The node is the removed (Section 2.2). If we compute the tokens as explained above, M and R must cooperate again to compute $h(k_D, l_{MRD})$. To avoid this it is necessary that the tuple owner Y agrees with each access requestor X on an initial key k_X^Y which will be used by the requestor as a starting point in the derivation process. Since we want that each database user remembers only one key we can set $k_X^Y = h(k_X, l_Y)$ where k_X is the unique private key of the requestor X.

4. Conclusions

This deliverable covers M4-M12 of Task 3.1 and provides a first version of the techniques for enforcing selective access on outsourced data. In particular, the deliverable illustrates an approach based on selective encryption, which is able to guarantee that data stored in the cloud self-enforce access control restrictions. The solution discussed in Chapter 2 can support data owners in the enforcement of both read and write privileges. The chapter also presents a solution that supports updates to access privileges without requiring expensive re-encryption operations, thus limiting the overhead for the data owner. The deliverable has also analyzed the collusion risks to which data are possibly exposed when grant and revoke operations modify the original access control policy. Chapter 3 specifically considered the problem of access control enforcement in the supply chain scenario (ESCUDO-CLOUD Use Case 2). To this purpose, it describes a solution based on RFIDs for the authentication of (authorized) players within a supply chain. It then presents how the selective encryption approach discussed in Chapter 2 can be used to enforce access restrictions in the supply chain scenario, when the collected data are stored in the cloud repository.

This deliverable provides the foundations for the techniques for selective access to the data that will be developed in the project. In particular it offers building blocks and concepts for the support of access restrictions, selective sharing requirements, and collaborative query processing that will be investigated in WP3. The analysis and results presented in the deliverable will also be considered for extending the tools for protection of data at rest developed within WP2 to the support of (possibly dynamic) sharing constraints.

Bibliography

[ABFF09] M. Atallah, M. Blanton, N. Fazio, and K. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Transactions on Information and System Security (TISSEC)*, 12(3):18:1–18:43, January 2009.

[AT83] S. Akl and P. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems (TOCS)*, 1(3):239–248, August 1983.

[BBD$^+$08] H. Bock, M. Braun, M. Dichtl, E. Hess, J. Heyszl, W. Kargl, H. Koroschetz, B. Meyer, and H. Seuschek. A milestone towards RFID products offering asymmetric authentication based on elliptic curve cryptography. In *Proc. of the 4th Workshop on RFID Security (RFIDSec 2008)*, Budapest, Hungary, July 2008.

[BGK$^+$07] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede. Public-key cryptography for RFID-tags. In *Proc. of 4th IEEE International Workshop on Pervasive Computing and Communication Security (PerSec 2007)*, New York, USA, March 2007.

[BJR$^+$06] J. Brainard, A. Juels, R. L. Rivest, M. Szydlo, and M. Yung. Fourth-factor authentication: Somebody you know. In *Proc. of the 13th ACM Conference on Computer and Communications Security (CCS 2006)*, Alexandria, VA, USA, October 2006.

[BLS04] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

[Bly99] P. Blythe. RFID for road tolling, road-use pricing and vehicle access control. In *IEE Colloquium on RFID Technology (Ref: No. 1999/123)*, London, 1999.

[CE09] S.-Y. Chou and Y. Ekawati. Cost reduction of public transportation systems with information visibility enabled by RFID technology. In *Proc. of the 16th ISPE International Conference on Concurrent Engineering (CE 2009)*, Taipei, Taiwan, July 2009.

[CMW06] J. Crampton, K. Martin, and P. Wild. On key assignment for hierarchical access control. In *Proc. of 19th IEEE Computer Security Foundations Workshop (CSFW 2006)*, Venice, Italy, July 2006.

[DEF$^+$14] S. De Capitani di Vimercati, R.F. Erbacher, S. Foresti, S. Jajodia, G. Livraga, and P. Samarati. Encryption and fragmentation for data confidentiality in the cloud. In A. Aldini, J. Lopez, and F. Martinelli, editors, *Foundations of Security Analysis and Design VII*. Springer, 2014.

[DFJ$^+$07] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Over-encryption: Management of access control evolution on outsourced data. In *Proc. of the 33rd International Conference on Very Large Data Bases (VLDB 2007)*, Vienna, Austria, September 2007.

[DFJ$^+$10] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Encryption policies for regulating access to outsourced data. *ACM Transactions on Database Systems (TODS)*, 35(2):12:1–12:46, April 2010.

[DFJ$^+$13] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati. Enforcing dynamic write privileges in data outsourcing. *Computers & Security*, 39:47–63, November 2013.

[DFJL12] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, and G. Livraga. Enforcing subscription-based authorization policies in cloud scenarios. In *Proc. of the 26th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec 2012)*, Paris, France, July 2012.

[DFM04] A. De Santis, A.L. Ferrara, and B. Masucci. Cryptographic key assignment schemes for any access control policy. *Information Processing Letters (IPL)*, 92(4):199–205, November 2004.

[DS08] B. L. Dos Santos and L. S. Smith. RFID in the supply chain. *Communications of the ACM*, 51(10):127–131, October 2008.

[DVW92] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, June 1992.

[EPC07] EPCglobal. EPCglobal architecture framework, Version 1.2, 2007.

[Fin03] K. Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*. John Wiley & Sons, Inc., 2003.

[Fuj08] Fujitsu. Fujitsu Develops World's First 64 KByte High-Capacity FRAM RFID Tag for Aviation Applications, 2008.

[FWR05] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES implementation on a grain of sand. *IEE Proceedings on Information Security*, 152(1):13–20, October 2005.

[GA07] E. Grummt and R. Ackermann. Proof of Possession: Using RFID for large-scale authorization management. In *Proc. of AmI-07 Workshop*, Darmstadt, Germany, November 2007.

[HWF08] D. Hein, J. Wolkerstorfer, and N. Felber. ECC is ready for RFID – A proof in silicon. In *Proc. of the 15th Annual Workshop on Selected Areas in Cryptography (SAC 2008)*, Sackville, New Brunswick, Canada, August 2008.

[KDSB09] F. Kerschbaum, D. Dahlmeier, A. Schröpfer, and D. Biswas. On the practical importance of communication complexity for secure multi-party computation protocols. In *Proc. of the 2009 ACM Symposium on Applied Computing (SAC 2009)*, Honolulu, HI, USA, March 2009.

[Ker08] F. Kerschbaum. Practical privacy-preserving benchmarking. In *Proc. of the 23rd IFIP International Information Security Conference (SEC 2008)*, Milan, Italy, September 2008.

[KOW10] F. Kerschbaum, N. Oertel, and L. Weiss Ferreira Chaves. Privacy-preserving computation of benchmarks on item-level data using RFID. In *Proc. of the 3rd ACM Conference on Wireless Network Security (WiSec 2010)*, Hoboken, NJ, USA, March 2010.

[KS09] F. Kerschbaum and A. Sorniotti. RFID-based supply chain partner authentication and key agreement. In *Proc. of the 2nd ACM Conference on Wireless Network Security (WiSec 2009)*, Zurich, Switzerland, March 2009.

[KTF08] C. Kuerschner, F. Thiesse, and E. Fleisch. An analysis of data-on-tag concepts in manufacturing. In *Proc. of the 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)*, Munich, Germany, February 2008.

[MSW05] D. Molnar, A. Soppera, and D. Wagner. A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags. In *Proc. of the 12th International Workshop on Selected Areas in Cryptography (SAC 2005)*, Santa Fe, New Mexico, USA, March 2005.

[NIS09] NIST. A survey of access control models. In *Privilege (Access) Management Workshop*, 2009.

[San87] R.S. Sandhu. On some cryptographic solutions for access control in a tree hierarchy. In *Proc. of the 1987 Fall Joint Computer Conference on Exploring Technology: Today and Tomorrow*, Dallas, TX, USA, October 1987.

[San88] R.S. Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters (IPL)*, 27(2):95–98, February 1988.

[SBE01] S.A Sarma, D. Brock, and D. W. Engels. Radio frequency identification and the electronic product code. *IEEE Micro*, 21(6):50–54, November 2001.

[Son08] B. Song. RFID tag ownership transfer. In *Proc. of the Workshop on RFID Security (RFIDSec 2008)*, Budapest, Hungary, July 2008.

[STF05] T. Staake, F. Thiesse, and E. Fleisch. Extending the EPC network: the potential of RFID in anti-counterfeiting. In *Proc. of the 2005 ACM Symposium on Applied Computing (SAC 2005)*, Santa Fe, New Mexico, USA, March 2005.

[SWE03] S. E. Sarma, S. A. Weis, and D. W. Engels. RFID systems and security and privacy implications. In *Proc. of the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002)*, San Francisco, CA, USA, August 2003.

[vDMV09] T. van Deursen, S. Mauw, and P Vullers. Secure ownership and ownership transfer in RFID systems. In *Proc. of the 14th European Symposium on Research in Computer Security (ESORICS 2009)*, Saint Malo, France, September 2009.

[WK08] L. Weiss Ferreira Chaves and F. Kerschbaum. Industrial privacy in RFID-based batch recalls. In *Proc. of the 1st IEEE International Workshop on Security and Privacy in Enterprise Computing (3M4EC 2008)*, Munich, Germany, September 2008.

[YT05] E. Yuan and J. Tong. Attributed based access control (ABAC) for web services. In *Proc. of the IEEE International Conference on Web Services (ICWS 2005)*, Orlando, FL, USA, July 2005.