Distributed Query Execution under Access Restrictions ⋆

Sabrina De Capitani di Vimercati[a], Sara Foresti[a], Sushil Jajodia[b], Giovanni Livraga[a], Stefano Paraboschi[c], Pierangela Samarati[a,*]

[a] *Università degli Studi di Milano*
Via Celoria 18, 20133 Milano (MI), Italy
firstname.lastname*@unimi.it*
[b] *George Mason University*
10401 York River Road, Fairfax, VA, 22030-4422, USA
jajodia@gmu.edu
[c] *Università degli Studi di Bergamo*
Viale Marconi 5, 24044 Dalmine (BG), Italy
parabosc@unibg.it

Abstract

The availability of a multitude of data sources has naturally increased the need for subjects to collaborate for supporting distributed computations that combine different data collections for their elaboration and analysis. Due to the quick pace at which datasets grow, often the authorities collecting and owning such datasets resort to external third parties (e.g., cloud providers) for their storage and management. Data under the control of different authorities are autonomously encrypted (using different encryption schemes and keys) for their external storage. This makes distributed computations combining these sources difficult to support. In this paper, we propose an approach enabling collaborative computations over data encrypted in storage, selectively involving also subjects that might not be authorized for accessing the data in plaintext when their collaboration is considered economically convenient. We also consider the possible adoption of trusted hardware components, to enable the evaluation of operations over plaintext data at non-fully trusted computational providers. The experimental results confirm the economic benefits that can be enabled by our proposal.

Keywords: Distributed query execution, Controlled data sharing, Authorization model, Relation profile, Cloud computing

1. Introduction

Our society and economy ever-increasingly rely on the knowledge that can be generated by analysis and computations combining data produced and/or controlled by different parties. The cloud, thanks to a

⋆A preliminary version of this paper appeared under the title "Distributed query evaluation over encrypted data," in *Proc. of the 35th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSec 2021)*, Calgary, Canada, July 19-20, 2021 [11]

*Corresponding author

variety of storage and computational providers with different costs and performance guarantees, represents an accelerator for such scenarios. Data owners can in fact outsource their data to storage providers, making them (selectively) available for computations with reduced management burden at their side. At the same time, users requiring analysis can (partially) delegate expensive computations to computational providers, with clear performance and economic benefits [9]. However, complications can arise since some of the data can be sensitive, proprietary, or more in general subject to access restrictions, all factors that can affect the possibility of relying on external cloud providers for data management and processing [15, 16, 22, 33].

To ensure data protection while permitting the consideration of a large spectrum of providers for computations, a recent approach proposed a simple, yet flexible, authorization model that enriches the traditional yes/no visibility that a subject can have over data with a third visibility level, granting a subject visibility over an encrypted version of the data [9]. In this way, subjects that are economically advantageous, but possibly not fully trusted for accessing data content, may still be involved in computations by restricting them to operate on encrypted data. To enforce the authorization policy, visibility over data is dynamically adjusted by applying encryption on-the-fly before sending data to subjects not trusted for plaintext access. Similarly, decryption can be applied on-the-fly when authorized subjects need plaintext visibility for executing query operations.

The authorization model in [9] operates under the assumption that the datasets involved in the distributed computation are stored in plaintext. This assumption is however viable only when data are either stored at their owners, or outsourced at providers that are trusted to access data in plaintext, hindering the consideration of storage providers that, while being economically convenient, are not authorized to see the plaintext content of the data. Intuitively, the spectrum of potential providers that could be adopted for storing datasets could be enlarged if data were encrypted by their owners, before outsourcing them for storage. The joint adoption of the authorization model in [9] and of encrypted storage would benefit both users requiring computations, and owners wishing to make their data selectively available to others. Economically convenient providers can then be leveraged by users for computation, and by owners to outsource their datasets with the guarantee that their data will not be improperly exposed in storage or in computation. The consideration of data encrypted in storage in collaborative computations brings however complications, since the encryption adopted for protecting data in storage is not specifically selected according to the computations to be performed and may not support them, which could hence require additional decryption and re-encryption operations.

In this paper, we build on the authorization model in [9] and propose a solution for collaborative computations over distributed data that can be stored, in encrypted form, at external and possibly not fully trusted providers. The main contributions of this paper can be summarized as follows. First, we re-define the information flows enacted by a computation, necessary for authorization enforcement, based on the possibility of some data being stored in encrypted form. Second, we identify the need, and propose a so-

lution for, re-encryption operations, to be introduced when the encryption adopted in storage (which is pre-determined by the data owner) does not support operation execution. Third, we provide an approach for computing an economically convenient assignment of operations to subjects in complete obedience of authorizations and provide an experimental evaluation demonstrating the economic benefits that can be enabled by our approach. Finally, we discuss the integration of trusted hardware components in our model to enable plaintext computations within the trusted boundaries of the trusted hardware made available by possibly non fully trusted providers.

The remainder of this paper is organized as follows. Section 2 discusses related works. Section 3 introduces the relation profiles modeling information flows entailed by computations, and presents the authorization model. Section 4 illustrates how authorizations are compared against relation profiles to determine when a subject is authorized for performing an operation, and formulates the problem of determining a minimum cost assignment of operations in a query plan to subjects. Section 5 presents a heuristic approach for solving such minimum cost assignment problem. Section 6 illustrates experimental results. Section 7 extends our model to the use of trusted hardware components. Finally, Section 8 concludes the paper. The proofs of the theorems are reported in Appendix A.

2. Related work

The work closest to ours is represented by the solution for distributed query evaluation in the respect of access restrictions proposed in [9], on which our approach builds. The model in [9] introduces the idea of specifying different visibility levels over data, including an encrypted visibility level, to the aim of enabling the delegation of computations over encrypted data to non-fully trusted subjects. This authorization model has been extended in [10] to queries including also set and rename operators, and to the consideration of the encryption cost in the allocation of operations to subjects. This model has been integrated into a real world query optimizer in [13]. The work in [9, 10] is based on the assumption that base relations are stored on the premises of the authorities owning them. Hence, base relations are available in plaintext and can be selectively encrypted on-the-fly, as needed to protect data visibility in query evaluation. Our proposal extends such an approach to consider the more general scenario where base relations might be stored, in encrypted form, at an external provider that might be not authorized to know the plaintext data. The consideration of encrypted storage has been first investigated in [11]. In this paper, we considerably extend the work in [11] by enriching its analysis, both theoretically and experimentally, by providing advanced considerations on the properties enjoyed by candidate sets along the query plan, and by extending the proposal to the adoption of trusted hardware components.

Techniques aimed at the management of distributed computations proposed in the literature (e.g., [2, 4, 19, 21, 26]) do not take into consideration access restrictions. In the context of relational databases, solutions

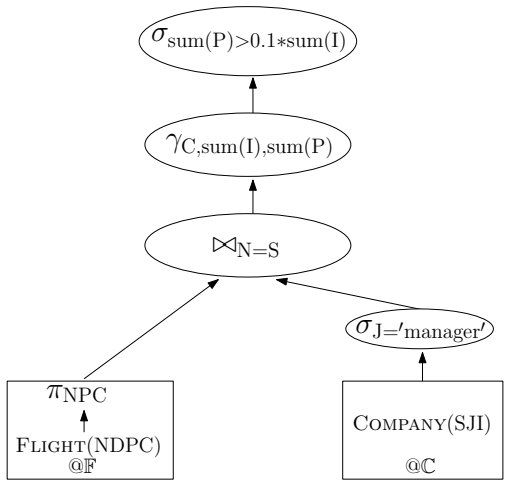
aimed at enforcing access restrictions (e.g., view-based access control [8, 17, 27], access patterns [3, 6], data masking [20]) instead do not consider encryption for protecting confidentiality.

Recent works have addressed the problem of protecting data confidentiality in distributed computation. The proposed solutions aim at controlling (explicit and/or implicit) information flows among subjects in the context of distributed computations (e.g., [12, 23, 28, 34]). The approach in [12] regulates implicit information flows due to joins among relations in the authorizations, and differs from ours as it requires collaborative specification of authorizations. The solution in [23] instead focuses on computations in hybrid clouds, and aims at limiting leakage of sensitive information to the public untrusted components of the clouds. The proposal in [34] considers different join execution strategies in distributed query evaluation, but it does not consider implicit information flows. The approach in [28] aims at controlling information flows for enforcing privacy constraints in operator placement for distributed query processing. This solution leverages on programming language techniques for providing privacy while maximizing performance. On a related line of work, in [14] the authors propose a solution aimed at protecting the confidentiality of the intents of a query to the providers involved in the evaluation of the query itself.

The use of encryption for protecting data confidentiality, while supporting query evaluation, has been widely studied (e.g., [1, 18, 24, 31]). Alternative solutions proposed the adoption of secure multiparty computation and homomorphic encryption (e.g., [5, 7]). These approaches are complementary to our work, which can rely on these techniques for delegating the evaluation of operations in the query plan (e.g., conditions and/or computations over attributes) to subjects who are authorized only for encrypted visibility over (a subset of) the attributes involved in the delegated operation. Recently, the use of trusted execution environments has been investigated for (distributed) computations over sensitive data (e.g. [25, 32, 30, 29]). These solutions require the presence of trusted hardware components for protecting data in computations over them. Our approach is complementary to those proposal, and is more general. In fact, our approach can leverage trusted hardware components for enabling computations over plaintext data at untrusted providers, but it does not require them, and is therefore applicable also to more traditional scenarios that are not equipped with trusted execution environments.

3. Relation profiles and authorizations

We consider a scenario characterized by: *i) data authorities*, each owning one or more relations possibly stored at external *storage providers*; *ii) users*, submitting queries over relations under the control of different authorities; and *iii) computational providers*, which can be involved in query evaluation. Since relations may be stored at external providers, (a subset of) their attributes might be encrypted by the data authority, to prevent visibility of sensitive information by the storage provider. Queries can be of the general form "SELECT FROM WHERE GROUP BY HAVING" and can include joins among relations under the control of



(a) Query plan

[NCP,_]→𝕌
[SJI,_]→𝕌

[PC,N]→𝕏
[_,SJI]→𝕏

[DP,CN]→𝕐
[JI,S]→𝕐

[NC,P]→ℤ
[S,JI]→ℤ

[_,NDPC]→𝔽
[_,JI]→𝔽

[_,NC]→ℂ
[_,SJI]→ℂ

[NC,_]→any
[_,SJ]→any

(b) Authorizations

Figure 1: An example of a query plan (a) and of authorizations on relations Flight and Company (b)

different data authorities. Execution of queries is performed according to a query plan where projections are pushed down to avoid retrieving data that are not necessary for query evaluation. Graphically, we represent query plans as trees whose leaf nodes correspond to projection over the base relations of the attributes involved in the query. For simplicity, but without loss of generality, we assume that attributes involved the relations have different names.

Example 3.1. *Consider two data authorities, a flight company and a commercial company, with one relation each, respectively: relation* Flight(N,D,P,C) *reports the social security Number and Date of birth of passengers, and the Price and Class of their tickets; relation* Company(S,J,I) *reports the Social security number, Job, and Income of the company employees. These relations are stored in encrypted form at external storage providers* $\mathbb{F}$ *and* $\mathbb{C}$*, respectively. We consider also three computational providers* $\mathbb{X}$*,* $\mathbb{Y}$*, and* $\mathbb{Z}$*. In our running example, we consider the following query submitted by user* $\mathbb{U}$*: "*select C, sum(P), sum(I) from Flight join Company on N=S where J='manager' group by C having sum(P)> 0.1*sum(I)*", retrieving the classes for which the overall price of tickets bought by managers is greater than the 10% of the summed incomes of the buyers. Figure 1(a) illustrates a plan for the execution of the query. For simplicity, in the figure and in the following, we denote a set of attributes with the sequence of the attributes composing it, omitting the curly brackets and commas (e.g.,* NPC *represents* {N,P,C}*).*

Relation profile. Besides the attributes included in its schema, a relation resulting from a computation might also convey information on other attributes. The information content explicitly and implicitly con-

veyed by a (base or derived, that is, resulting from the evaluation of a sub-query) relation is captured by the *profile* of the relation. We extend the definition of relation profile in [9] to account for the possible encrypted representation of attributes encrypted in storage.

Definition 3.1 (Relation Profile). *Let R be a relation. The* profile *of R is a 6-tuple of the form $[R^{vp}, R^{ve}, R^{vE}, R^{ip}, R^{ie}, R^{\simeq}]$ where: R^{vp}, R^{ve}, and R^{vE} are the* visible *attributes appearing in R's schema in plaintext (R^{vp}), encrypted on-the-fly (R^{ve}), and encrypted in-storage (R^{vE}); R^{ip} and R^{ie} are the* implicit *attributes conveyed by R, in plaintext (R^{ip}) and encrypted (R^{ie}); $R^{\simeq}$ is a disjoint-set data structure representing the closure of the equivalence relationship implied by attributes connected in R's computation.*

In the definition, R^{vp} corresponds to the set of plaintext attributes visible in the schema of R. We then distinguish between the visible attributes encrypted on-the-fly (R^{ve}) and the visible attributes encrypted in storage (R^{vE}), due to their different nature. In-storage encryption is enforced once, independently from the query to be answered, with an encryption scheme and a key that do not change over time and known only to the data owner. On-the-fly encryption is enforced at query evaluation time and both the encryption scheme and the encryption key are decided by the user formulating the query and can be shared among different parties when different attributes need to be compared (e.g., for a join evaluation). Implicit components (R^{ip}, R^{ie}) keep track of the attributes that have been involved in query evaluation for producing relation R. Even if they do not appear in R's schema, query evaluation has left a trace of their values in the query results (e.g., attributes involved in selection or group by operations). Note that we do not distinguish between in-storage and on-the-fly encryption in the implicit component of the profile. Indeed, the information leaked by the evaluation of an operation over an encrypted attribute is not influenced by the time at which encryption has been enforced nor by the subject enforcing it. The equivalence relationship ($R^{\simeq}$) keeps track of the sets of attributes that have been compared for query evaluation (e.g., for the evaluation of an equi-join). The component allows accounting for the fact that attributes involved in some comparison can leak one the values of the other. In fact, even if one of the attributes in the equivalence set has been projected out from the relation schema, its values can still be inferred from other (equivalent) attributes.

The profile of a *base* relation R has all components empty, except R^{vp} and R^{vE} that contain the attributes appearing in plaintext and encrypted in storage, respectively, in the schema of the relation. The profile of a *derived* relation (i.e., resulting from the evaluation of an operation) depends on both the operation and the profile of the operand(s). Figures 2 and 3 illustrate the profiles resulting from the evaluation of relational algebra operators (Figure 2), and of encryption, decryption, and re-encryption (i.e., decryption followed by encryption with a different scheme and/or key) operations (Figure 3), which are peculiar of our model. For each operator, we report the general formula on the left hand side of the figures, and an example on the right hand side of the figures. Graphically, we represent the profile of a relation as a tag attached to the relation's node (or the node of the operator producing it in case of a derived relation), with three components:

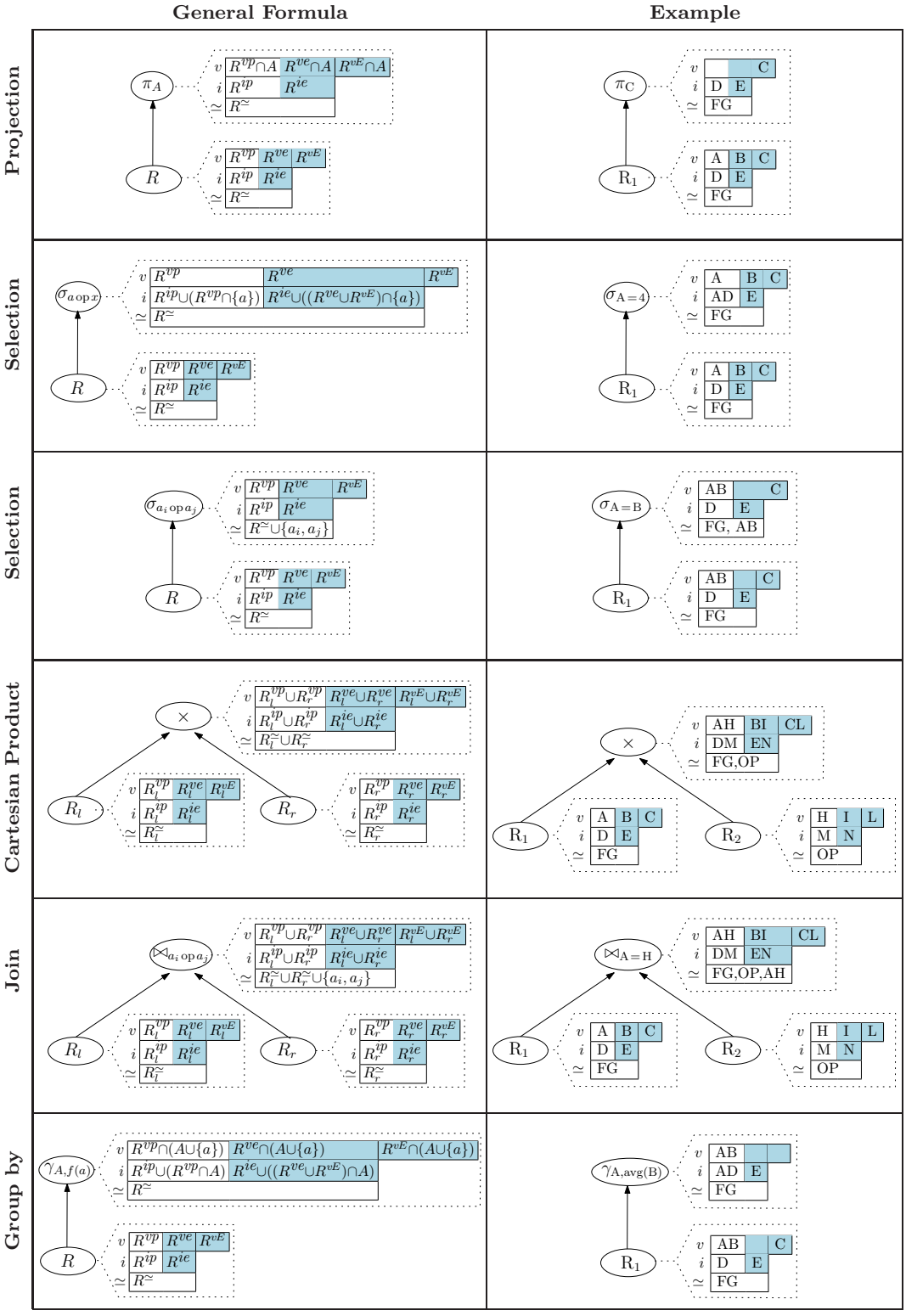


Figure 2: Profiles resulting from relational operations

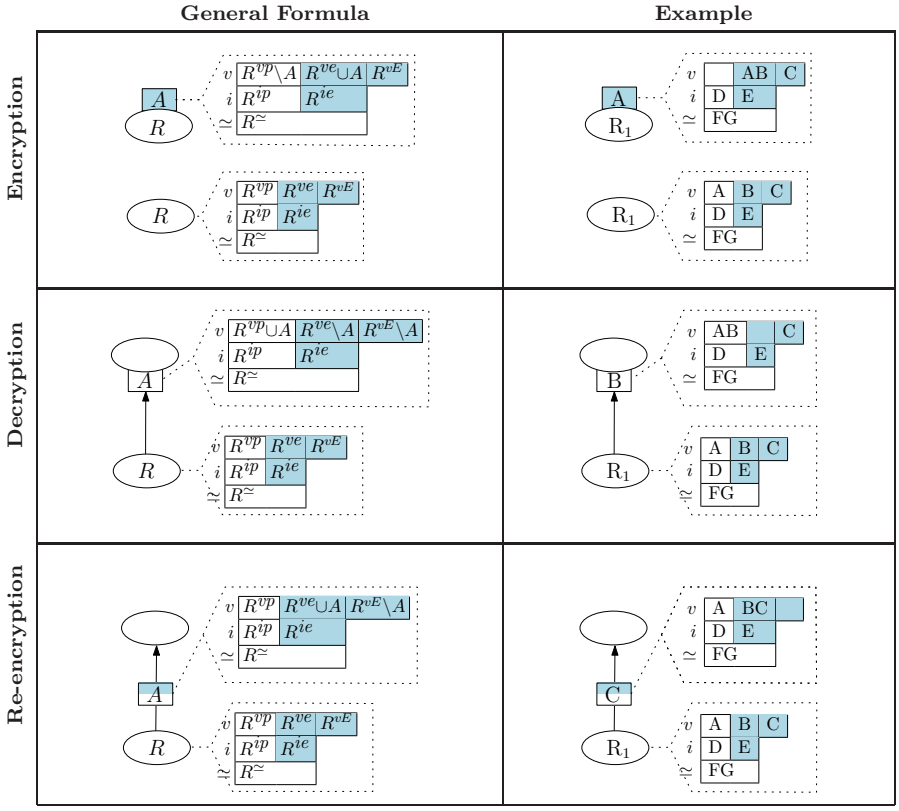


Figure 3: Profiles resulting from encryption, decryption, and re-encryption operations

v (visible attributes in R^{vp} and, on a light blue background, R^{ve} and R^{vE}), i (implicit attributes in R^{ip} and, on a light blue background, R^{ie}), and $\simeq$ (sets of equivalent attributes in $R^{\simeq}$ that have been compared for R's computation). We represent encryption and decryption operations as light blue nodes and white nodes, respectively, containing the attributes to be encrypted/decrypted, attached to the operand or the resulting relation, respectively. Re-encryption operations are represented as light blue and white nodes containing the attributes to be re-encrypted. In the following, we illustrate those components of relation profiles that are impacted by the evaluation of relational algebra operators and of encryption, decryption, and re-encryption. As it is visible from Figures 2 and 3, projection π_A maintains in the visible components of the resulting relation profile only the attributes in A. A selection of the form $\sigma_{a \,\mathrm{op}\, x}$ includes attribute a in the implicit component of the resulting profile (plaintext or encrypted, depending on whether a is plaintext or encrypted for the selection). A selection of the form $\sigma_{a_i \,\mathrm{op}\, a_j}$, comparing attributes a_i and a_j, inserts $\{a_i, a_j\}$ in the equivalence set of the resulting profile. Cartesian product $\times$ merges each of the components of its operands in the resulting relation profile. Similarly, join $\bowtie_{a_i \,\mathrm{op}\, a_j}$ merges each of the components of its operands and, requiring the comparison between a_i and a_j, adds $\{a_i, a_j\}$ to the equivalence set of the resulting profile. Group by $\gamma_{A,f(a)}$, grouping the input relation by a set A of attributes and evaluating an aggregate function f

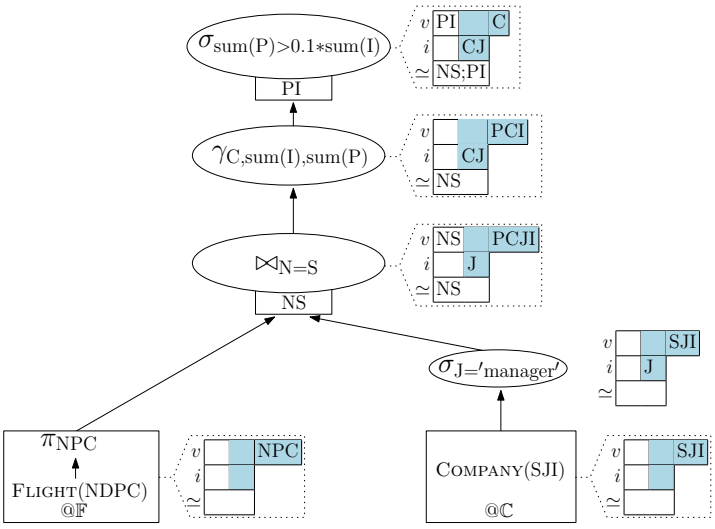


Figure 4: Query plan with profiles

on an attribute a, maintains only a and attributes in A in the visible components of the resulting profile, and adds A to the implicit (encrypted or plaintext, depending on whether A is encrypted or plaintext) component of the resulting profile. Encryption and decryption move attributes from the visible plaintext to the visible encrypted component (and vice-versa) of the profile of the resulting relation. Re-encryption moves attributes from the encrypted in-storage component to the encrypted component. Figure 4 illustrates the profiles of the relations resulting from the evaluation of the operations in the query plan in Figure 1(a), assuming attributes NS and PI to be decrypted for enabling computations over them.

Authorizations. Authorizations regulate data flows intended for computations. Authorizations can specify, for each subject, whether she has plaintext visibility, encrypted visibility, or no visibility for performing computations over the attributes in the relations, and are defined as follows.

Definition 3.2 (Authorization). *Let R be a relation and $\mathcal{S}$ be a set of subjects. An* authorization *is a rule of the form $[P, E]\rightarrow S$, where $P\subseteq R$ and $E\subseteq R$ are subsets of attributes in R such that $P\cap E=\emptyset$, and $S\in\mathcal{S}\cup\{any\}$.*

Authorization $[P, E]\rightarrow S$ states that subject S can access in plaintext attributes in P, in encrypted form attributes in E, and has no visibility over the attributes in $R\backslash(P\cup E)$. Subject 'any' can be used to specify a default authorization applying to all subjects for which no authorization has been explicitly defined. Authorizations regulating access for computation over (encrypted) attributes in relation R are defined by the data authority who owns the relation, independently from the provider storing it. Note that the authorizations of storage providers depend on whether they are to be considered also for computations, independently from the fact that they store a specific relation and its (encrypted or plaintext) form. The user

formulating a query is expected to have plaintext visibility over a subset of the attributes in the relational schemas, and we assume that she is authorized for the attributes involved in the query.

Example 3.2. *Figure 1(b) illustrates an example of authorizations regulating access to relations* Flight *and* Company *of Example 3.1. User* $\mathbb{U}$ *has plaintext visibility over a subset of the attributes of the two relations, storage providers* $\mathbb{F}$ *and* $\mathbb{C}$ *have encrypted visibility over the attributes in the relation they store, and encrypted visibility over some of the attributes of the other relation, computational providers* $\mathbb{X}$, $\mathbb{Y}$, *and* $\mathbb{Z}$ *have either plaintext or encrypted visibility over a subset of the attributes in the two relations.*

Authorization verification. To be considered authorized to view a relation, a subject needs plaintext visibility over plaintext attributes (R^{vp} and R^{ip}) and plaintext or encrypted visibility over encrypted attributes (R^{ve}, R^{vE}, and R^{ie}). Note that there is no need to distinguish between in-storage and on-the-fly encryption for authorization verification, as the information conveyed by encrypted attributes is independent from the time at which encryption has been applied. The subject also needs to have the same visibility (plaintext or encrypted) over attributes appearing together in an equivalence set. This is required to prevent subjects having plaintext visibility on one attribute in the equivalence set and encrypted visibility on another to be able to exploit knowledge of plaintext values of the former to infer plaintext values of the latter.

In the following, for simplicity, we will denote with $\mathcal{P}_S$ ($\mathcal{E}_S$, respectively) the set of attributes that a subject S can access in plaintext (encrypted, respectively) according to her authorizations. The following definition identifies subjects authorized to access a relation, extending the definition in [9] to take the two kinds of encryption into consideration.

Definition 3.3 (Authorized Relation). *Let* R *be a relation with profile* $[R^{vp},R^{ve},R^{vE},R^{ip},R^{ie},R^{\simeq}]$. *A subject* $S \in \mathcal{S}$ *is* authorized *for* R *iff:*

1. $R^{vp} \cup R^{ip} \subseteq \mathcal{P}_S$ (*authorized for plaintext*);
2. $R^{ve} \cup R^{vE} \cup R^{ie} \subseteq \mathcal{P}_S \cup \mathcal{E}_S$ (*authorized for encrypted*);
3. $\forall A \in R^{\simeq}$, $A \subseteq \mathcal{P}_S$ *or* $A \subseteq \mathcal{E}_S$ (*uniform visibility*).

Example 3.3. *Consider a relation* R *with profile* [P,C,S,_,_,{IP}] *and the authorizations in Figure 1(b). Provider* $\mathbb{Z}$ *is not authorized for* R *since it cannot access* P *in plaintext (Condition 1);* $\mathbb{C}$ *and* $\mathbb{F}$ *are not authorized for* R *since they cannot access* P *and* S, *respectively, in any form (Condition 2);* $\mathbb{X}$ *is not authorized since it does not have uniform visibility on* P *and* I *(Condition 3). Provider* $\mathbb{Y}$ *and user* $\mathbb{U}$ *are instead authorized for* R.

For simplicity, in the following we will use notation R_i to denote the relation resulting from the evaluation of node n_i in the query tree plan. When clear from the context, we will use n_i to denote interchangeably the node and the corresponding relation.

4. Extended minimum cost query plan

Given a query tree plan, denoted $\mathtt{T(N)}$, corresponding to a query q formulated by a user $\mathbb{U}$, our goal is to determine, for each node, a subject for its evaluation, possibly extending the query plan with encryption, decryption, and re-encryption operations to guarantee the satisfaction of authorizations and enable the evaluation of operations.

4.1. Candidates

Given a query tree plan $\mathtt{T(N)}$, we first need to identify, for each node, the subjects authorized for evaluating it (i.e., its candidates). Given a node n in a query tree plan, a subject S is authorized for its execution if she is authorized for its operand(s) and for its result. Indeed, S needs to access the operands of the node for its evaluation, and the profile of the result captures all the information directly and indirectly conveyed by the evaluation of the operation. Starting from relations where attributes may be encrypted for storage, it could be necessary to inject decryption and re-encryption (i.e., decryption followed by encryption with a different scheme and/or key) to guarantee that operations can be evaluated when they require plaintext visibility over the involved attributes, or they are not supported by the encryption scheme adopted in storage, respectively. For instance, we cannot expect different data authorities to use the same encryption scheme and key for attributes that will be compared in an equi-join. Hence, even if equality conditions can easily be supported over encrypted data (e.g., using deterministic encryption), the evaluation of equi-joins requires re-encryption of the join attributes. Besides decryption and re-encryption for enabling query evaluation, also encryption operations could be injected for enforcing authorizations: encryption could enable a subject to perform an operation that she would otherwise not be authorized to evaluate, due to the plaintext representation in the operand relation(s) of some attributes that she can access only in encrypted form.

Example 4.1. *With reference to the query tree plan in Figure 4,* $\mathbb{Y}$ *can evaluate the join operation if attributes* N *and* S *are re-encrypted using a deterministic encryption scheme with the same encryption key. Similarly, attributes* P*,* I*, and* J *must be encrypted, all with the same key, for* $\mathbb{Z}$ *to be authorized to execute the group by operation.*

We observe that, if all the attributes in the schema of the operand relation(s) appear in encrypted form, the set of subjects that are authorized for evaluating the operation is possibly larger. In fact, encrypted attributes are also accessible by subjects with plaintext visibility (Definition 3.3). To determine candidates for operation execution, we therefore assume that all the attributes in the operand relation(s), but those that have to be in plaintext for operation execution, are encrypted. Any attribute of the operand(s) can be decrypted by the subject who is in charge for the evaluation of the operation, since otherwise it would not be authorized to evaluate it. Formally, we define candidates for the evaluation of a node as follows.

Definition 4.1 (Candidate). *Let* $\mathtt{T(N)}$ *be a query tree plan,* $n \in \mathtt{N}$ *be a non-leaf node, and* $n_l, n_r \in \mathtt{N}$ *be its left child and right child (if any),* $n.A_p$ *be the set of attributes that need to be plaintext for evaluating* n*, and* $\mathcal{S}$ *be a set of subjects. A subject* $S \in \mathcal{S}$ *is a* candidate *for the execution of a node* n *iff* S *is authorized for:*

1) n_l *and* n_r*, assuming the encryption of all the visible attributes (Definition 3.3);*

2) attributes in $n.A_p$ *in plaintext;*

3) n*, assuming the encryption of all the visible attributes in its operand(s) (Definition 3.3).*

The set of candidates for node n *is denoted* $\Lambda(n)$*.*

Example 4.2. *Figure 5(a) reports, for each node in the query tree plan in Figure 4, the candidates that can evaluate the operation represented by the node. In the example, we assume that:* i) *the selection over* J *and the computation of the sums over* I *and* P *can be evaluated on their encrypted in-storage representation;* ii) *the evaluation of the join and of the group by clause over* C *require the re-encryption of the involved attributes; and* iii) *the comparison of* SUM(P) *and* SUM(I) *can be performed on plaintext values only.*

A query tree plan $\mathtt{T(N)}$ complemented with encryption, decryption, and re-encryption operations represents an *extended query plan*, and is denoted $\mathtt{T'(N')}$. Encryption, decryption, and re-encryption operations are inserted into the query plan to enforce authorizations and ensure operation execution. The injection of operations that change attribute visibility (i.e., encryption and decryption) depends on the subject selected, among all the candidates, for the execution of each operation in the query plan. To enable the evaluation of operation at node n, which requires plaintext visibility over encrypted attribute a, we inject decryption of a as a child node attached to n. Similarly, to hide visibility over plaintext attribute a to the subject in charge of evaluating n, but which can access a only in encrypted form, we inject encryption of a as a parent node attached to the child of n including a in its profile. We note that encryption and decryption operations can always be enforced by the subject evaluating the operation to which they are attached (i.e., the child node where the attribute appears in plaintext for encryption, and the parent node that requires plaintext visibility on the attribute for decryption). For this reason, we do not need to explicitly compute the candidates for encryption and decryption nodes (and we do not report candidates for such nodes in our figures). The consideration of re-encryption operations, necessary when the in-storage encryption scheme does not support operation execution, deserves a special treatment. To enable the evaluation of the operation, not supported by in-storage encryption, at node n over attribute a, we inject re-encryption of a in the subtree rooted at n. Re-encryption is not necessarily injected as a child of n. In fact, it might be that neither the subject evaluating n nor the subject in charge of the evaluation of its operand are authorized for it. If this is the case, re-encryption needs to be enforced at a lower level in the query plan tree. As an example, re-encryption of a can be injected as the parent of the leaf node representing the base relation

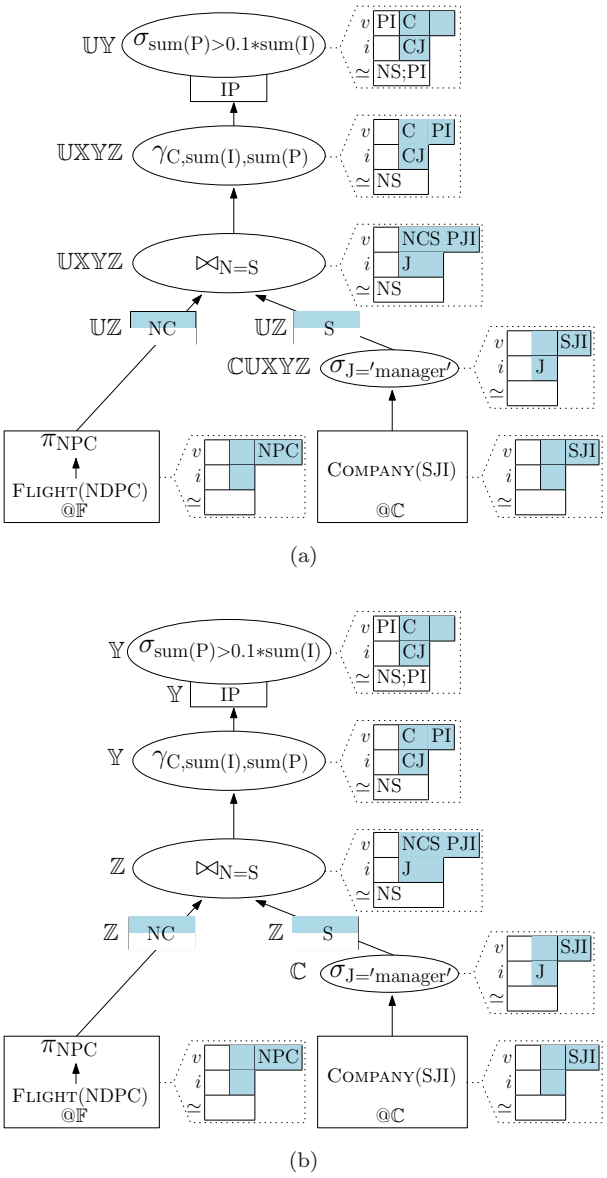


Figure 5: Extended query plan with candidates (a) and with assignees (b)

to which a belongs. Another difference with respect to encryption and decryption operations, is that the need for injecting a re-encryption operation depends on the operations in the query plan (and not on the subject to which they are assigned), based on whether they are supported by the in-storage encryption. For the reasons above, for re-encryption operations, it is necessary to reason on (and hence define) the set of subjects that can perform them. The definition of candidates therefore applies also to re-encryption nodes.

Example 4.3. *Figure 5(a) illustrates an example of an extended version of the query tree plan in Figure 4, where attributes* N, C, *and* S *are re-encrypted, and attributes* I *and* P *are decrypted. The decryption of attributes* I *and* P *can be performed by either* $\mathbb{U}$ *or* $\mathbb{Y}$*, which are candidates of the selection operating on*

the decryption result and can access I *and* P *in plaintext. Figure 5(a) reports, for the two re-encryption operations, the set of candidates that are authorized to re-encrypt the involved attributes. Note that the re-encryption of attribute* S *cannot be assigned to* $\mathbb{C}$ *(to which the selection* $\sigma_{\mathrm{J}='\mathrm{manager}'}$ *could be assigned), nor to* $\mathbb{X}$ *or* $\mathbb{Y}$ *(to which the join could be assigned). Indeed, none of these subjects can access* S *in plaintext (see authorizations in Figure 1(b)). It could instead be assigned to* $\mathbb{Z}$*.*

The sets of candidates associated with the nodes of a query tree plan enjoy a nice *monotonicity* property, according to which the candidates of a node are a subset of the candidates of its descendants in the query tree plan. This property applies to all nodes representing operations that are executed over encrypted attributes or that need to be executed over plaintext attributes and leave a trace of such attributes in the plaintext implicit component of the resulting relation profile (i.e., $n.A_p \subseteq R^{\dot{ip}}$).

The consideration of re-encryption operations introduces a local gap in the monotonicity property. Since the subject in charge of re-encryption must be authorized for the profile of the operand relation, the set of candidates for a re-encryption operation is a subset of the candidates of its operand node n_c. However, the candidates of the parent n_p of the re-encryption operation might not be a subset of the re-encryption candidates. In fact, nothing can be said on the set containment relationship between re-encryption candidates and those of its parent n_p, since a candidate for re-encryption could not be authorized for n_p and vice-versa: while a subject must be authorized for plaintext visibility on the attributes to be re-encrypted to be candidate for re-encryption, n_p might not require (and its candidate might not have) plaintext visibility on these attributes. Similarly, the evaluation of n_p might require plaintext visibility over attributes that do not need to be visible in plaintext for re-encryption. The profile of the result of re-encryption is the same as the one of its operand (i.e., it does not move attributes from the encrypted to the plaintext components nor vice-versa). Note that the set of candidates for n_p is a subset of the set of candidates for n_c, since a candidate for n_p needs to have at least visibility on the relation produced by n_c.

Example 4.4. *With reference to the extended query tree plan in Figure 5(a), the set of candidates for the nodes in the original query tree plan decreases while going up in the tree. Consider the node for the re-encryption of attribute* S*. The set of candidates for this node is a subset of the set of candidates for the selection operation, child of re-encryption (i.e.,* $\{\mathbb{U},\mathbb{Z}\} \subseteq \{\mathbb{C},\mathbb{U},\mathbb{X},\mathbb{Y},\mathbb{Z}\}$*). Also, the set of candidates for the re-encryption of* S *is a subset of the set of candidates for the join operation, parent of re-encryption (i.e.,* $\{\mathbb{U},\mathbb{Z}\} \subseteq \{\mathbb{U},\mathbb{X},\mathbb{Y},\mathbb{Z}\}$*, since only* $\mathbb{U}$ *and* $\mathbb{Z}$ *are authorized for plaintext visibility over* S*).*

Formally, the monotonicity property among the sets of candidates is stated by the following theorem.

Theorem 4.1 (Monotonicity of the candidate set). *Let* $\mathtt{T}(\mathtt{N})$ *be a query tree plan and* $\mathtt{N}^{\mathtt{r}}$ *the set of re-encryption nodes injected into* $\mathtt{T}(\mathtt{N})$ *to enable query evaluation.* $\forall n_x, n_y \in \mathtt{N} \cup \mathtt{N}^{\mathtt{r}}$ *such that* n_x,n_y *are non-leaf nodes of* $\mathtt{T}(\mathtt{N})$*,* n_y *is a child of* n_x*, and* $n_y.A_p \subseteq R_x^{\dot{ip}}$*:*

1. $\Lambda(n_x) \subseteq \Lambda(n_y)$, *if* n_x, $n_y \in \mathtt{N}$;
2. $\Lambda(n_z) \subseteq \Lambda(n_y)$, *if* $n_x \in \mathtt{N}^{\mathtt{r}}$ *and* $n_z \in \mathtt{N}$ *is the parent of* n_x.

Given a query tree plan $\mathtt{T}(\mathtt{N})$ and the candidates for each of the nodes in $\mathtt{N}$ and for re-encryption operations, it is necessary to select, for each node and re-encryption operation, a subject (chosen among its candidates) in charge of the evaluation of the corresponding operation (i.e., the assignee $\lambda(n)$ of the node n representing the operation). There can exist different possible assignments that respect authorizations and permit query execution. In the next sub-section, we discuss how to determine an authorized assignment.

4.2. Authorized assignment and minimum cost query plan

Given a query tree plan $\mathtt{T}(\mathtt{N})$ and the set $\Lambda(n)$ of candidates for each node $n \in \mathtt{N}$, our goal is to determine an assignment of nodes to subjects taken from the corresponding set of candidates. This can require to inject encryption, decryption, and re-encryption operations in $\mathtt{T}(\mathtt{N})$ (i.e., generating an extended query plan $\mathtt{T}'(\mathtt{N}')$ for $\mathtt{T}(\mathtt{N})$) to enforce authorizations and ensure that operations can be computed. An assignment of nodes to subjects exists if *i)* each node has at least a candidate and, *ii)* for each attribute a that needs to be re-encrypted, there exists a subject who can access a in plaintext and the other attributes in the schema of the base relation to which a belongs in encrypted or plaintext form. Encryptions are inserted to enforce authorizations and attached to the child of the node assigned to a subject authorized for encrypted visibility only. Decryptions are inserted to adjust attributes visibility for operation evaluation, and are attached to the node requiring plaintext visibility for operation evaluation. As discussed in Section 4.1, these operations can be performed by the same subject as that assigned to the nodes to which they are attached, and hence the assignee of an encryption or a decryption operation is the assignee of the node on which it operates. Re-encryption, on the contrary, could be assigned to a different subject (as discussed in Section 4.1, a re-encryption operation has its own candidates and may be assigned to neither the assignee of its parent, nor to the assignee of its child). We also note that re-encryption operations could be inserted at any point in the query plan, in the subtree rooted at the node that represents the operation for which re-encryption is needed. We recall that the need for re-encryption of an attribute a does not depend on the choice of assignments, but only on: *i)* the in-storage encryption (scheme and key) of a; and *ii)* the operations to be evaluated over a for query execution. Hence, independently from the selected assignment, if no subject has plaintext visibility over a and encrypted (or plaintext) visibility over all the other attributes in the base relation to which a belongs, there cannot exist any authorized assignment for the query plan. On the contrary, if such a subject exists, there is at least an authorized assignment for the query plan. Indeed, the re-encryption operation can be evaluated as early as when the relation leaves the storage provider.

Example 4.5. *Consider attribute* C *of Example 3.1, which needs to be re-encrypted for the evaluation of the* GROUP BY *clause in Figure 1(a). An authorized assignment requires a subject that can access attributes*

N *and* P *in encrypted form and* C *in plaintext for re-encryption of* C. *Since* $\mathbb{U}$, $\mathbb{X}$, *and* $\mathbb{Z}$ *can access* N *and* P *encrypted and* C *plaintext, in the worst case scenario, re-encryption of* C *can be injected as a parent of the leaf node representing base relation* FLIGHT *and can be assigned to one among* $\mathbb{U}$, $\mathbb{X}$, *and* $\mathbb{Z}$.

The existence of an authorized assignment is formalized by the following theorem.

Theorem 4.2 (Existence of an authorized assignment). *Let* $\mathtt{T}(\mathtt{N})$ *be a query tree plan,* $\mathcal{S}$ *be a set of subjects, and* $\forall n \in \mathtt{N}$, $n.A_e$ *be the set of attributes that need to be re-encrypted for the evaluation of* n *and* $\Lambda(n)$ *be the set of candidates for* n. *If* $\forall n \in \mathtt{N}$, $\Lambda(n) \neq \emptyset$ *and,* $\forall a \in n.A_e$, *there exists at least a subject* $S \in \mathcal{S}$ *s.t.* $a \in \mathcal{P}_S$ *and* $R \subseteq \mathcal{P}_S \cup \mathcal{E}_S$, *with* R *the base relation to which* a *belongs, then there exists at least an extended query plan* $\mathtt{T}'(\mathtt{N}')$ *of* $\mathtt{T}(\mathtt{N})$ *and an assignment* $\lambda : \mathtt{N}' \rightarrow \mathcal{S}$ *of subjects to nodes in* $\mathtt{T}'(\mathtt{N}')$ *such that:*

1. $\lambda(n) = \lambda(n_p)$, *with* n_p *the parent of* n, *if* n *is a decryption operation;*
2. $\lambda(n) = \lambda(n_c)$, *with* n_c *the child of* n, *if* n *is an encryption operation;*
3. $\lambda(n) \in \Lambda(n)$, *otherwise;*

that does not violate any authorization.

We can then conclude that, if there exists a set of candidates for each node of a query tree plan, any combination of subjects chosen from the sets of candidates of the nodes in the query plan can be made authorized by injecting encryption, decryption, and re-encryption operations. Indeed, injecting re-encryption of all the attributes encrypted in-storage as parent of leaf nodes permits to make any assignment selected from the candidate set authorized simply by adjusting attributes visibility with encryption/decryption operations, as demanded by authorizations and operations evaluation (see [9]). For instance, Figure 5(b) illustrates an extended query plan that makes the assignment on the left of each node authorized according to the authorizations in Figure 1(a).

Among the possible assignments, we expect the user formulating the query to be interested in selecting the one that optimizes performance, economic costs, or both of them. In the considered cloud scenario, we expect the economic cost to be the driving factor in the choice of the candidates. The economic cost for the evaluation of a query includes two main factors: *i) computational cost* for the evaluation of the operations in the query tree plan; and *ii) data transfer cost* for the relations exchanged between subjects for query evaluation.

The cost of query evaluation is obtained by summing the computational and data transfer costs, taking into consideration also encryption, decryption, and re-encryption operations. Formally, the problem of computing an assignment that minimizes the cost of query evaluation is formulated as follows.

Problem 4.1 (Minimum cost query tree plan). *Let* $\mathtt{T}(\mathtt{N})$ *be a query tree plan and* $\mathcal{S}$ *be a set of subjects. Determine an extended query plan* $\mathtt{T}'(\mathtt{N}')$ *of* $\mathtt{T}$ *and an assignment* $\lambda : \mathtt{N}' \rightarrow \mathcal{S}$ *such that:*

MAIN($\mathtt{T}(\mathtt{N})$, $\mathcal{S}$)

1: **Compute_Cost**($\mathtt{T}$.*root*) /* Step 1: pre-compute costs (Figure 7) */
2: insert a node *client* as parent of $\mathtt{T}$.*root* assigned to the user $\mathbb{U}$ formulating the query
3: **Identify_Candidates**($\mathtt{T}$.*root*) /* Step 2: identify candidates (Figure 8) */
4: *to_enc_dec*=$\emptyset$
5: **Compute_Assignment**($\mathtt{T}$.*root*) /* Step 3: compute assignment and inject re-encryption operations (Figure 9) */
6: **Extend_Plan**($\mathtt{T}$.*root*) /* Step 4: inject encryption/decryption operations (Figure 10) */

Figure 6: Pseudocode of our heuristic algorithm

1. $\forall n \in \mathtt{N}'$: $\lambda(n) = \lambda(n_p)$, *with* n_p *the parent of* n, *if* n *is a decryption operation;* $\lambda(n) = \lambda(n_c)$, *with* n_c *the child of* n, *if* n *is an encryption operation;* $\lambda(n) \in \Lambda(n)$, *otherwise;*
2. $\forall n \in \mathtt{N}'$, $\lambda(n)$ *is authorized for the profiles of* n *and of its children;*
3. $\nexists \mathtt{T}''$, λ' *such that* $\mathtt{T}''$ *is an extended query plan of* $\mathtt{T}$ *and* λ' *an assignment for* $\mathtt{T}''$ *such that* $\forall n \in \mathtt{N}$, $\lambda'(n) \in \Lambda(n)$ *and* $\text{cost}(\mathtt{T}'', \lambda') < \text{cost}(\mathtt{T}', \lambda)$, *with* $\text{cost}(\mathtt{T}', \lambda)$ ($\text{cost}(\mathtt{T}'', \lambda')$, *resp.) the cost of evaluating* $\mathtt{T}'$ *with assignment* λ *(of evaluating* $\mathtt{T}''$ *with assignment* λ', *resp.).*

The problem of computing a minimum cost query plan is hard. We therefore propose a heuristic approach for its solution.

5. Computing assignment

In this section, we illustrate our heuristic approach for computing an assignment (and the corresponding extended query tree plan) that satisfies authorizations, while minimizing the economic cost. For simplicity, but without loss of generality, in the description and in the pseudo-code, we assume all attributes in base relations to be encrypted in storage. The algorithm receives as input a query tree plan $\mathtt{T}(\mathtt{N})$ and the set $\mathcal{S}$ of subjects that the user is willing to involve in query evaluation with their authorizations. It computes an extended query tree plan and an assignment. We illustrate our heuristic approach in Section 5.1 and analyze its correctness and complexity in Section 5.2.

5.1. Heuristic approach for computing a minimum cost query tree plan

The proposed heuristics operates in four steps (see Figure 6). The first step, corresponding to a pre-processing, pre-computes the computation costs which would result when assigning sub-trees of the input query plan to each possible subject in $\mathcal{S}$. The second step identifies the set of candidates associated with the nodes of the query plan. The third step chooses, for each operation in the query plan, the subject (among the corresponding candidates) in charge of its execution, and inserts re-encryption operations as needed. The last step inserts encryption and decryption operations. The procedures corresponding to these steps are

Compute_Cost(n)

1: **if** $n_l \neq$ NULL **then Compute_Cost**(n_l)
2: **if** $n_r \neq$ NULL **then Compute_Cost**(n_r)
3: **for each** $S \in \mathcal{S}$ **do**
4: $\quad$ *comp_cost*[n,S] = *comp_cost*[n_l,S] + *comp_cost*[n_r,S] + n.*comp_cost*·S.*comp_price*

Figure 7: Pseudocode of procedure **Compute_Cost**

reported in Figures 7, 8, 9, and 10 and described in the following. In the discussion and in the procedures, given a node n, we denote with n_p its parent, and with n_l and n_r its left and right child, respectively.

Compute costs. Recursive procedure **Compute_Cost** (Figure 7) visits the query plan in post-order and pre-computes the cost of the evaluation of the subtree rooted at n, assuming to assign the entire subtree to subject S, for each node n in the query plan and each subject S. The computed costs are stored in a matrix having a row for each node of the query plan and a column for each subject. Value *comp_cost*[n,S] represents the cost of evaluating the subtree rooted at n at S. To precompute costs, the procedure performs a post-order visit of the query plan. Starting from the leaves, for each node n, the procedure computes *comp_cost*[n,S] summing the cost of the evaluation of the subtrees rooted at the children of n at S with the cost of evaluating n at S, for each subject S. Note that, when visiting n, the costs *comp_cost*[n_l,S] and *comp_cost*[n_r,S] of evaluating the left and right children of n at S have already been computed. The cost of evaluating n at S is instead computed by multiplying the estimated computational complexity of evaluating the operation represented by n by the computational price of S.

Identify candidates. Recursive procedure **Identify_Candidates** (Figure 8) performs a post-order visit of the query tree plan to identify, for each node, the candidates for its evaluation. To this end, for each node n, the procedure needs to compute its profile. For leaf nodes (lines 4–6), all components are empty but $n.vE$, which contains all the attributes in the schema of the base relation that survive the first (pushed-down) projection (Section 3). The procedure then initializes the set of candidates for the leaf nodes to the entire set $\mathcal{S}$ of subjects of the system (line 7): this is done to simplify the computation of the candidates of the other nodes in the tree (clearly, leaf nodes are assigned to the storage provider storing the corresponding base relation, regardless of the candidates identified at this step). For non-leaf nodes, according to Definition 4.1, the procedure computes the candidates assuming that all the attributes in the operands are encrypted, unless demanded for the evaluation of n (lines 9–13). To compute the candidates of a non-leaf node n (lines 16–22), the procedure leverages the monotonicity property among candidate sets (Theorem 4.1) and checks, for each of the candidates of its children, whether the subject is also a candidate for n. When the operations at the child(ren) require plaintext visibility on attributes (i.e., $n_l.A_p \cup n_r.A_p \neq \emptyset$) and those attributes did not leave a trace in the profile of n (i.e., $n_l.A_p \cup n_r.A_p \not\subseteq n.ip$), the algorithm searches for candidates among all subjects.

Identify_Candidates(n)

```
1:  if n_l≠NULL then Identify_Candidates(n_l)
2:  if n_r≠NULL then Identify_Candidates(n_r)
3:  /* compute the profile of the node over its (encrypted) children */
4:  if n_l=n_r=NULL /* n is a leaf node */
5:  then  n.vp=n.ve= n.ip= n.ie= n.eq=∅
6:            n.vE=R /* all the attributes in the relation schema are encrypted */
7:            Λ(n)=S /* any subject */
8:            n.TotA_p=n.TotA_e=∅
9:  else  let n.A_p be the set of attributes that need to be plaintext for evaluating n
10:           let n.A_e be the set of attributes that need to be (re)encrypted on-the-fly for evaluating n
11:           n_l=encrypt(n_l−n.A_p, decrypt(n.A_p∪ n.A_e, n_l))
12:           n_r=encrypt(n_r−n.A_p, decrypt(n.A_p∪ n.A_e, n_r))
13:           Compute_Profile(n) /* compute the relation profile according to Figure 2 */
14:           n.TotA_p=n.A_p∪n_l.TotA_p∪n_r.TotA_p
15:           n.TotA_e=n.A_e∪n_l.TotA_e∪n_r.TotA_e
16:           Λ(n)=∅
17:           if n_l.A_p∪n_r.A_p⊆n.ip
18:           then Cand=Λ(n_l) ∪ Λ(n_r)
19:           else Cand=S
20:           for  each S∈Cand do
21:                if  S is authorized for n_l, n_r, n
22:                then Λ(n)=Λ(n)∪{S}
23:           if Λ(n)=∅ then exit
```

Figure 8: Pseudocode of procedure **Identify_Candidates**

If no subject is a candidate for n, the procedure terminates (line 23) since n cannot be evaluated without violating at least an authorization. Procedure **Identify_Candidates** also sets variables $n.TotA_p$ ($n.TotA_e$, resp.) to the set of attributes that must be plaintext (encrypted on the fly, resp.) for the evaluation of the subtree rooted at n (lines 8, 14–15).

Compute assignment. Recursive procedure **Compute_Assignment** (Figure 9) performs a pre-order visit of the query tree plan. Intuitively, for each visited node, the procedure chooses between assigning the evaluation of the node to the same subject as its parent n_p (without paying any transfer cost), and moving it to a different subject, if economically convenient. Economic convenience is evaluated comparing the cost of evaluating the whole subtree rooted at n at each subject S candidate of the node (note that, as illustrated in Theorem 4.1, if a subject S is a candidate for a node n in the original query plan, it is also a candidate for all its descendants in the plan). To estimate the cost of delegating the evaluation of the subtree rooted at n to S, we consider the following cost components.

- *Data transfer cost* (lines 21–22) applies only when n is assigned to a subject S different from its parent, and is the cost of transferring the relation generated by n from S to the subject in charge of n_p. Data transfer cost is computed as the product between the estimated size of the relation generated by n

Compute_Assignment(n)

```
1:  S_min=NULL
2:  min=+∞
3:  if n_l=n_r=NULL /* n is a leaf node */
4:  then λ(n)=n.S /* storage provider for the corresponding relation */
5:       if to_enc_dec∩R≠∅ /* R includes attributes to be re-encrypted */
6:       then att=to_enc_dec∩R
7:            Cand=S
8:            while att≠∅ ∧ Cand≠∅ do
9:                 let S∈Cand be the subject with minimum (comp_price+transf_price)
10:                Cand=Cand\{S}
11:                dec=att∩P_S
12:                if dec≠∅ ∧ R⊆(P_S∪E_S)
13:                then insert a re-encryption node new as parent of n for dec
14:                     λ(new)= S
15:                att=att\dec
16:           if att≠∅
17:           then exit /* at least an attribute cannot be re-encrypted without violating authorizations */
18: /* n is an internal node */
19: else if n is not a re-encryption operation
20:      then for each S∈Λ(n) do
21:           if S≠λ(n_p) then cost=n.size·S.transf_price /* transfer cost */
22:           else cost=0 /* no transfer cost */
23:           cost = cost+comp_cost[n,S] /* computational cost */
24:           for each a∈(n.TotA_p∪n.TotA_e)∩P_S do /* S decrypts the attribute */
25:                cost=cost+dec_cost(a)·S.comp_price
26:           for each a∈(n.TotA_e\P_S) do /* need to delegate re-encrypt of a */
27:                cost = cost+(dec_cost(a)+enc_cost(a))·avg_comp_price+
28:                       a.size(avg_transf_price+S.transf_price)
29:           for each a∈(to_enc_dec∩P_S) do /* S can re-encrypt a */
30:                cost=cost+(dec_cost(a)+enc_cost(a))·S.comp_price
31:           if cost<min
32:           then min=cost
33:                S_min=S
34:      /* select the subject in charge of the evaluation of n */
35:      λ(n)= S_min
36:      if to_enc_dec∩P_λ(n)≠∅
37:      then insert a re-encrypt node new for to_enc_dec∩P_λ(n) as parent of n
38:           λ(new)=λ(n)
39:           to_enc_dec=to_enc_dec\P_λ(n)
40:      to_enc_dec=to_enc_dec∪(n.A_e\P_λ(n)) /* delegated re-encryption */
41:      if n.A_e∩P_λ(n)≠∅
42:      then insert a re-encrypt node new for n.A_e∩P_λ(n) as child of n
43:           λ(new)=λ(n)
44: if n_l≠NULL then Compute_Assignment(n_l)
45: if n_r≠NULL then Compute_Assignment(n_r)
```

Figure 9: Pseudocode of procedure **Compute_Assignment**

and the transfer price of the subject S in charge of evaluating n (in line with cloud market price lists, we consider only outbound traffic).

- *Computational cost* (line 23) models the costs entailed by the evaluation of the operations in the subtree rooted at n by S. It is computed as the sum of the costs (pre-computed by procedure **Compute_Cost** and stored in matrix *comp_cost*) of evaluating all the nodes in the subtree rooted at n by subject S.

- *Decryption cost* (lines 24–25) is the cost of decrypting the attributes that need to be plaintext (or encrypted on-the-fly) for the evaluation of n or one of its descendants (i.e., any node in the subtree rooted at n that S is in charge of evaluating). The decryption cost is estimated by multiplying the decryption cost of each attribute a by the computational price of S.

- *Re-encryption cost* (lines 26–30) includes the cost of re-encryption operations performed by S as well as of re-encryption operations necessary for the evaluation of n but that need to be delegated to a different subject since S is not authorized for plaintext visibility on (a subset of) these attributes. To keep track of the attributes that require re-encryption, we use variable *to_enc_dec*, which lists the attributes that require re-encryption for the evaluation of the ancestors of n. If S can access a subset of the attributes in *to_enc_dec* in plaintext, the algorithm assigns its re-encryption to S (lines 29–30). If S needs to operate on an attribute a encrypted on-the-fly on which she does not have plaintext visibility, S clearly cannot decrypt a, which is then to be re-encrypted by another subject. In this case, the algorithm estimates the cost of injecting a re-encryption operation into the query plan, performed by a third party authorized for it. Such a cost is estimated as the sum of the costs for encrypting and decrypting the attribute of interest (assuming the average computational price of the subjects in the system), and the transfer cost for sending the relation to the subject in charge of re-encryption and then back to S (lines 26–28).

Among the candidates for node n, procedure **Compute_Assignment** selects the subject S_{min} with minimum estimated cost (line 33). Depending on the chosen assignee $\lambda(n)$, the procedure injects re-encryption operations and updates variable *to_enc_dec*: $\lambda(n)$ is assigned the re-encryption of attributes in *to_enc_dec* that it is authorized to access in plaintext, and these attributes are removed from *to_enc_dec* (lines 36–39). Attributes in $n.A_e$ that $\lambda(n)$ cannot access in plaintext are instead inserted into *to_enc_dec*, to push re-encryption down in the query plan (line 40). Attributes in $n.A_e$ that $\lambda(n)$ can access in plaintext are re-encrypted by $\lambda(n)$. To this purpose, the algorithm injects a re-encryption operation, assigned to $\lambda(n)$, as a child of n (lines 41–43). Note that $\lambda(n)$ can decide to decrypt the attributes that need to be re-encrypted before evaluating n, and encrypt them (on the fly) after the evaluation of n. Since re-encryption operations are already assigned to a subject upon injection in the tree, further recursive calls of procedure **Compute_Assignment** do not need to operate on them (**if** condition, line 19).

Leaf nodes deserve a special treatment, since they do not represent operations and can only be assigned to the provider storing the corresponding base relation (lines 3–4). However, when the visit reaches a leaf node, it is necessary to verify whether all needed re-encryption operations have already been injected in the tree or if any re-encryption operation has been pushed down to the leaves. In this case, it is necessary to insert, as parent of the leaves, a set of re-encryption operations to ensure that the query can be executed. When visiting a leaf, the procedure checks whether the base relation R represented by the leaf includes attributes appearing in *to_enc_dec* (line 5). If this is the case, the procedure checks the subjects in increasing order of computational and transfer prices. It assigns to a subject the re-encryption of all the attributes it can access in plaintext, in the attempt of limiting encryption costs. The procedure then inserts, for each subject S selected for a re-encryption operation, a re-encryption node as parent of the leaf, assigned to S and operating on all the attributes for which S has been selected (lines 6–15). Clearly, if there is an attribute a to be re-encrypted and no subject authorized to re-encrypt it, the entire procedure terminates (lines 16–17). We note that the need to involve a subject for these re-encryption operations happens only if no subject assigned to the other operations in the query plan can access in plaintext the attribute(s) that need to be re-encrypted.

Extend plan. Recursive procedure **Extend_Plan** (Figure 10) performs a post-order visit of the query plan to inject encryption and decryption operations as needed. For the root node, the procedure injects a decryption of the encrypted attributes in the root (lines 3–5). For each non-root node n, the procedure injects a decryption node for each of n's children if n requires plaintext visibility over attributes that are encrypted in the profiles of its children. These decryption nodes are injected as parents of n's children, and are assigned to $\lambda(n)$ (lines 6–13). The procedure then computes the profile of n (line 14). Lastly, the procedure injects an encryption node for the attributes that are plaintext in n's profile but that can only be accessed in encrypted form by the assignee $\lambda(n_p)$ of n's parent n_p. This encryption node is injected as parent of n, and is assigned to $\lambda(n)$ (lines 15–18).

Example 5.1. *Considering the query tree plan and authorizations in Figure 1, the algorithm first visits the tree in post-order and identifies the candidates for each node (Figure 5(a)). The algorithm then visits the tree in pre-order and selects, for each node, the candidate that is more promising from an economic point of view (Figure 5(b)). For instance, assuming that* $\mathbb{Y}$ *is less expensive, the root node is assigned to* $\mathbb{Y}$*. Similarly, we assume that evaluating the* GROUP BY *clause at* $\mathbb{Y}$ *is more convenient than moving it to* $\mathbb{X}$ *or* $\mathbb{Z}$*. However, since* $\mathbb{Y}$ *cannot access attribute* $\mathrm{C} \in n.A_e$ *in plaintext,* C *is inserted into set* to_enc_dec *and its re-encryption pushed down in the tree. Assuming that the less expensive alternative for join evaluation is* $\mathbb{Z}$*, since* $\mathbb{Z}$ *can re-encrypt* C*, a re-encryption operation for* C *is inserted into the tree as child of the join node. Also, since both* S *and* N *need to be re-encrypted for the evaluation of the join operation and* $\mathbb{Z}$ *is authorized for this operation,* $\mathbb{Z}$ *decrypts and re-encrypts also* S *and* N*. We note that* $\mathbb{Z}$ *can evaluate the*

Extend_Plan(n)

```
1:  if n_l≠NULL then Extend_Plan(n_l)
2:  if n_r≠NULL then Extend_Plan(n_r)
3:  if n=T.root
4:  then  insert a decryption node new for n.ve∪n.vE as parent of n
5:        λ(new)=𝕌
6:  else  if n_l≠NULL AND (n.A_p∩(n_l.vp ∪ n_l.ve))\n_l.vp≠∅
7:        then  insert a decryption node new_l for (n.A_p∩(n_l.vp ∪ n_l.ve))\n_l.vp as parent of n_l
8:              Compute_Profile(new_l)
9:              λ(new_l)=λ(n)
10:       if n_r≠NULL AND (n.A_p∩(n_r.vp ∪ n_r.ve))\n_r.vp≠∅
11:       then  insert a decryption node new_r for (n.A_p∩(n_r.vp ∪ n_r.ve))\n_r.vp as parent of n_r
12:             Compute_Profile(new_r)
13:             λ(new_r)=λ(n)
14:       Compute_Profile(n)
15:       if E_λ(n_p)∩n.vp≠∅
16:       then  insert an encryption node new_e for E_λ(n_p)∩n.vp as parent of n
17:             Compute_Profile(new_e)
18:             λ(new_e)=λ(n)
```

Figure 10: Pseudocode of procedure **Extend_Plan**

join over plaintext values, being authorized for such visibility, and encrypt their values before sending the join result to $\mathbb{Y}$*. Finally, we assume that the selection over* J *can be evaluated over the attribute encrypted in storage and is then evaluated by the provider storing relation* COMPANY *(i.e.,* $\mathbb{C}$*). The third step of the algorithm injects encryption and decryption operations as needed: in the example, the decryption of* P *and* I *by* $\mathbb{Y}$ *for the evaluation of the root node.*

5.2. Complexity and correctness analysis

In this section, we analyze the complexity and correctness of the algorithm in Figure 6.

Theorem 5.1 (Complexity). *Let* $\mathtt{T}(\mathtt{N})$ *be a query tree plan,* $\mathcal{A}$ *be the set of attributes in the base relations of the plan, and* $\mathcal{S}$ *be the set of subjects. The complexity of the algorithm in Figure 6 is* $O(|\mathtt{N}|\cdot|\mathcal{S}|\cdot|\mathcal{A}|)$ *in time.*

Our heuristic approach guarantees that, if there exists an authorized assignment for the query tree plan, the algorithm finds it and generates the corresponding extended query plan. Formally, this is stated by the following theorem.

Theorem 5.2 (Correctness). *Let* $\mathtt{T}(\mathtt{N})$ *be a query tree plan and* $\mathcal{S}$ *be the set of subjects. If there exist an extended query tree plan* $\mathtt{T}'(\mathtt{N}')$ *of* $\mathtt{T}(\mathtt{N})$ *and an assignment* $\lambda : \mathtt{N}' \rightarrow \mathcal{S}$ *such that:*

1. $\forall n \in \mathtt{N}'$*:* $\lambda(n) = \lambda(n_p)$*, with* n_p *the parent of* n*, if* n *is a decryption operation;* $\lambda(n) = \lambda(n_c)$*, with* n_c *the child of* n*, if* n *is an encryption operation; and* $\lambda(n) \in \Lambda(n)$*, otherwise;*

2. $\forall n \in \mathtt{N}'$, $\lambda(n)$ *is authorized for the profile of* n *and of its children (Definition 3.3);*

the algorithm in Figure 6 terminates and finds it.

6. Experimental results

To test the economic benefits brought by our approach, we performed a set of experiments comparing the costs of executing its computed extended query plans with respect to a baseline, representing the costs of executing the original query plans at the user side. For our experiments, we considered a set of queries that is representative of a use-case, provided by a large manufacturing company that applies data analysis to extract information from production data combined with customers data and data provided by external agencies. The queries operate on four relations, stored in encrypted form at three storage providers. The query tree plans differ in the number of relations involved (ranging from 2 to 4), in the shape of the query plan, and in the number of nodes in the plans (ranging from 2 to 4) corresponding to operations whose execution requires the re-encryption of some attributes.

The cost parameters considered in the experimental evaluation have been derived from the price lists of the major cloud providers. The services offered by cloud providers have currently become quite varied, with increasingly complex price lists. Still, the major elements that contribute to the evaluation of the cost paid for the execution of queries remain the use of computational resources and the cost for the transfer of data, as captured by our proposal. With respect to the use of computational resources, a clear recent evolution is the greater flexibility in the determination of this cost: it is still common to evaluate the cost in terms of hours of use of a virtual machine with a given cpu/ram configuration, but some services offer the option to pay at finer granularity, even at the scale of the number of seconds that a given computational solution has been used. With respect to the transfer of data, the cost is in many cases linearly associated with the size of the data transmitted, often giving to customers free use of data bandwidth incoming into the cloud provider infrastructure. The determination of the cost parameters in the model will then have to consider the specific configuration and price list associated with the providers in the considered scenario. We note that the experiments do not focus on the absolute values, but on the ratio between the cost of resources offered by cloud providers and the cost of the baseline. Based on considerations from our use case and on the price lists of the most common cloud providers on the market, we set the cost values input to the experiments considering, as it is to be expected in our scenarios, a relatively high cost for the direct involvement of the user. In particular, we assumed the cpu usage and data transfer costs of the user to range from 10 (10x) to 100 (100x) times that of providers.

As for authorizations, which open the possibility of involving external providers in query execution, we considered two authorization configurations:

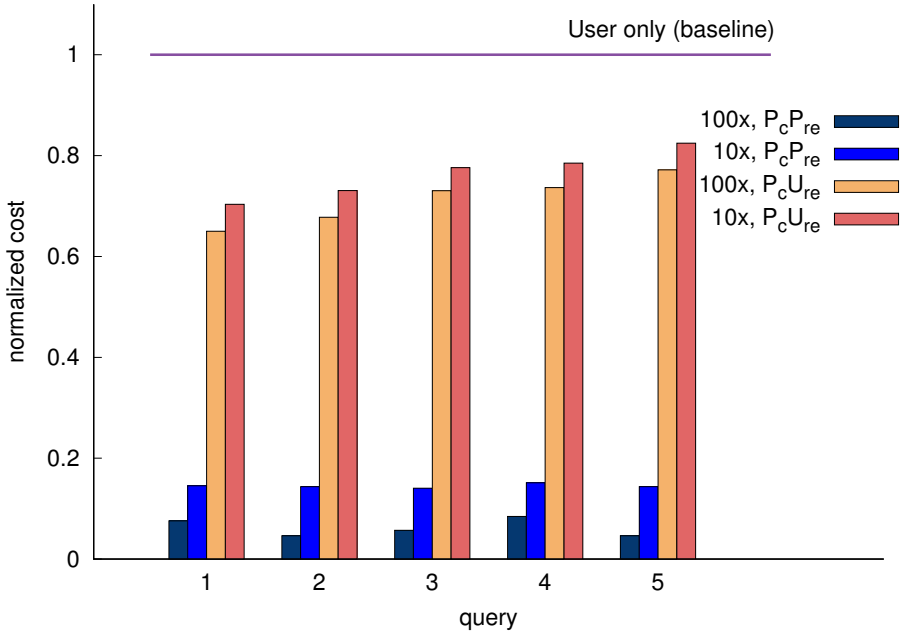


Figure 11: Normalized cost for evaluating different queries with different authorization configurations

- $\mathsf{P_cP_{re}}$, where external providers are trusted for operation execution (i.e., they have at least encrypted visibility over the involved attributes) as well as for re-encryption (i.e., they have plaintext visibility over the re-encrypted attributes)
- $\mathsf{P_cU_{re}}$, where external providers are trusted only for operation execution but cannot re-encrypt data, which is hence delegated to the user (who can then rely on providers for computation only).

Figure 11 illustrates the economic benefits, compared to the local execution at the user, considering the 10x and 100x cost configurations combined with the two authorization configurations (additional experiments considering intermediate configurations in the 10-100 range show similar results). The figure reports the costs in a normalized form assuming, as unitary cost for each query, local execution at the user. The figure shows that the involvement of cloud providers enables significant savings, since our approach permits to partially delegate computation, when economically convenient. When re-encryption is required, cost savings can increase if cloud providers can be involved, besides for computation, also for re-encryption ($\mathsf{P_cP_{re}}$). For all queries, the higher the cost ratios, the higher the economic benefits: the already significative savings in the 10x cost configuration (reaching 86% for q_3 in the $\mathsf{P_cP_{re}}$ authorization configuration) further increases in the 100x scenario (reaching 95% for q_2 and q_5 in the $\mathsf{P_cP_{re}}$ authorization configuration). The delegation of re-encryption operations to economically convenient providers enables larger savings (89% on average). There are, however, significant benefits also in the $\mathsf{P_cU_{re}}$ authorization configuration (in our use-case, the saving obtained involving the user for re-encryption reaches 35% for q_1, with an average of 26%).

7. Integration of trusted hardware components

Our approach for enforcing authorizations is based on the dynamic (on-the-fly) adjustment of the visibility over data to ensure that no information be improperly leaked to a subject involved in a computation. Plaintext data are then encrypted before being passed to a subject not trusted to access them in plaintext. Recently, many hardware producers have developed novel platforms equipped with trusted processor and secure storage space (e.g., Intel SGX), for enabling secure computation on the premises of non-fully-trusted providers. With reference to our scenario, the availability of trusted hardware components permits to rely on a non-fully trusted provider to evaluate operations over sensitive data in plaintext, even if the computational provider is not authorized for plaintext visibility over the involved attributes. Encryption and decryption operations, in fact, would take place within the boundaries of the trusted hardware, and the computational provider hosting it would not learn anything about the (plaintext) values over which the trusted hardware works. This possibility can be relevant to our problem, for example, when operations need plaintext visibility to be executed. In the remainder of this section, we illustrate how our approach can leverage the availability of trusted hardware components that may be available at some of the computational providers in $\mathcal{S}$. We first discuss how the use of trusted hardware components can be reflected in the specification of authorizations. Then, we discuss the impact on candidate definition and on profile computation.

Authorizations. Consider a provider S, equipped with a trusted hardware component. We expect authorizations regulating access to attributes for the trusted hardware component to be more permissive than the authorizations for the provider hosting it. For this reason, given a provider S equipped with a trusted hardware component, the trusted hardware should be treated as a different subject S_t, related to S but with its own authorizations. Clearly, the authorizations of S and of its trusted component S_t are not independent. For instance, if S can access an attribute in plaintext, the same attribute should be accessible in plaintext also to the (more trusted) component S_t. Given a relation R and the authorization $[P, E]\rightarrow S$ regulating access for S to R, and $[P_t, E_t]\rightarrow S_t$ the authorization regulating access for S_t to R, we expect the following conditions to hold.

1. $P \subseteq P_t$: the trusted hardware component can access in plaintext a superset of the plaintext attributes that the provider hosting it can access;
2. $(P_t \cup E_t) \subseteq (P \cup E)$: the trusted hardware component can access a subset of the attributes that the provider hosting it can access, independently from their representation.

Condition 1 ensures that S_t can access in plaintext at least the same attributes accessible to S. This is in line with the fact that S_t is considered more trusted than S. Condition 2 ensures that S_t cannot operate on attributes for which S is not authorized. This reflects the fact that the transmission of data to the trusted hardware component is mediated by the provider hosting it. To illustrate, consider an attribute a such that

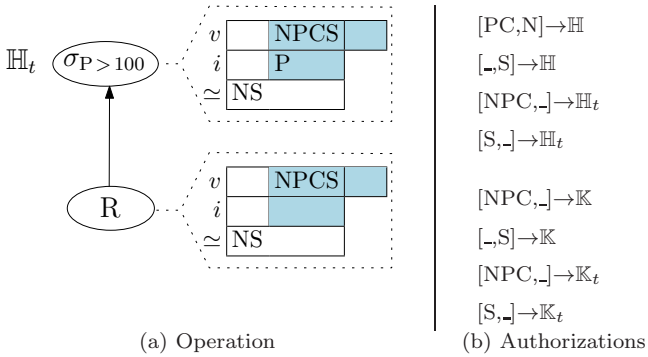
$a \in P_t \cup E_t$ and $a \notin P \cup E$ (i.e., S_t is authorized to access a in plaintext or encrypted form, while S can access a in neither plaintext nor encrypted form). Even though S_t is authorized to access a, the possibility for S_t to be involved in a computation over a would be prevented by the fact that, being S_t hosted at S, a would still need to pass through S to be delivered to S_t, a possibility ruled out by the fact that S is not authorized for a in any form. Hence, authorizing a trusted hardware component for an attribute over which the hosting subject does not have visibility would not bring any benefit.

Example 7.1. *Consider a provider* $\mathbb{J}$ *equipped with a trusted hardware component* $\mathbb{J}_t$*, and authorizations* [_,N]→$\mathbb{J}$ *and* [NP,_]→$\mathbb{J}_t$*. While satisfying Condition 1, these authorizations do not satisfy Condition 2. In particular, attribute* P *can be accessed only by* $\mathbb{J}_t$*. For this reason, no relation including in its profile attribute* P *could be assigned to* $\mathbb{J}_t$*, since it would disclose* P *also to the provider* $\mathbb{J}$ *hosting* $\mathbb{J}_t$*, violating authorizations.*

Candidates. The fact that transmission of data to the trusted hardware component is mediated by the provider hosting it has an impact on the definition of candidates. In particular, to determine whether S_t is a candidate for the evaluation of a node n of a query plan, checking whether S_t satisfies Definition 4.1 (like it is done for regular subjects) is not sufficient. It is also necessary to verify whether the flow of information passing through S, and entailed by the evaluation of n at S_t, is authorized. In other words, when checking whether S_t is a candidate for n, it is also necessary to check whether S can access the input to n (which S receives from n's children and passes to S_t for computation) as well as the output of n (which S receives from S_t and passes to n's parent). S must be authorized for such profiles assuming that all the visible attributes appear encrypted: trusted hardware component can decrypt the attributes that need to be plaintext for the evaluation of n, and re-encrypt them when generating the result.

Example 7.2. *Consider the selection over attribute* P *in Figure 12(a), and the authorizations in Figure 12(b) for subjects* $\mathbb{H}$ *and* $\mathbb{K}$*, and for their hosted trusted hardware components* $\mathbb{H}_t$ *and* $\mathbb{K}_t$*. While being authorized for both the input and the output profile of the selection, the trusted component* $\mathbb{K}_t$ *cannot be considered a candidate for the selection, since its hosting subject* $\mathbb{K}$ *is not authorized for the profile of the input relation (*$\mathbb{K}$ *does not have uniform visibility over attributes* N *and* S*). Indeed, since the input relation would be passed to* $\mathbb{K}_t$ *by* $\mathbb{K}$*, this would entail an unauthorized information flow. On the contrary,* $\mathbb{H}_t$ *is a candidate for the selection, since exposing the input and output relations to the hosting subject* $\mathbb{H}$ *would not entail unauthorized information flows.*

Profiles. Even if S_t operates on plaintext data, provider S hosting it (as well as any cloud provider) can observe anything on the operations executed by S_t, thanks to the security guarantees of the trusted hardware component. In fact, data are locally decrypted at S_t, and no information can leak from it. For this reason, operations executed at S_t do not leave a plaintext trace (as it instead happens with regular subjects



(a) Operation (b) Authorizations

Figure 12: An example of an operation assigned to a trusted hardware component (a) and of authorizations (b)

when executing operations on plaintext data) in the profile of the resulting relation. In other words, the profile of the result of operations evaluated by trusted hardware components are obtained assuming that the operations are evaluated over encrypted data (even if the trusted hardware component decrypts the attributes for computation and then re-encrypts them).

Example 7.3. *Consider the selection over attribute* P *in Figure 12(a) and assume that it is assigned to trusted hardware component* $\mathbb{H}_t$*, which is authorized for* P *in plaintext. Even if* $\mathbb{H}_t$ *performs the selection over plaintext values,* P *is decrypted and re-encrypted in a transparent way for its hosting subject* $\mathbb{H}$*.* $\mathbb{H}$ *would in fact receive the result of the selection after the re-encryption by* $\mathbb{H}_t$*, without plaintext information flows. Hence, as illustrated in the figure, the profile of the resulting relation includes* P *in the implicit encrypted component, in the same way as if the selection were computed over encrypted data.*

From the discussion above, it is clear that trusted hardware components, while requiring some care in the specification of authorizations and in the evaluation of candidates and profiles, can successfully be integrated in our model for distributed query evaluation, permitting computations over plaintext data without entailing unauthorized plaintext information flows.

We close this section with a consideration on the impact that the adoption of trusted hardware components can have on the economic cost of query evaluation since we expect such costs to be a driving factor for delegating computations to external and non-fully trusted subjects. The price lists applied by computational providers for using trusted hardware components tend to be higher than the cost for using regular hardware. On the contrary, we do not expect differences in terms of data transfer costs. We also note that, even if the trusted and regular hardware of a subject are physically hosted on two different servers, we do not expect any data transfer cost implied by the exchange of information between them, as data never leave the premises of the computational provider.

8. Conclusions

We proposed an approach for leveraging storage and computational providers to enable distributed query execution involving data possibly stored in encrypted form. Our solution permits collaborative query execution, selectively involving computational providers to reduce the cost of query execution, while ensuring obedience of authorizations. The proposed heuristics aims at limiting the economic cost of query evaluation by choosing, for each node, the candidate that is locally more economically convenient. The experimental evaluation confirms that our approach provides economic advantages to users who can leverage external providers for (distributed) query evaluation. We also investigated the involvement of computational providers offering a trusted hardware component for the evaluation of operations over sensitive data in plaintext, relying on the security guarantees provided by the trusted hardware component.

Acknowledgements

This work was supported in part by the Office of Naval Research under grant N00014-20-1-2407, by the Army Research Office under grant W911NF-13-1-0421, by the National Science Foundation under grant CNS-1822094, by the EC under projects MARSAL and GLACIATION, by the Italian MUR under PNRR project SERICS and PRIN project HOPE, and by JPMorgan Chase & Co under project "k-anonymity for AR/VR and IoT/5G".

References

[1] R. Agrawal, D. Asonov, M. Kantarcioglu, and Y. Li. Sovereign joins. In *Proc. of ICDE*, Atlanta, GA, USA, April 2006.

[2] W. Alkowaileet, S. Alsubaiee, M. Carey, C. Li, H. Ramampiaro, P. Sinthong, and X. Wang. End-to-end machine learning with Apache AsterixDB. In *Proc. of DEEM*, Houston, TX, USA, June 2018.

[3] A. Amarilli and M. Benedikt. When can we answer queries using result-bounded data interfaces? In *Proc. of PODS*, Houston, TX, USA, June 2018.

[4] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia. Spark SQL: Relational data processing in Spark. In *Proc. of SIGMOD*, Melbourne, Australia, May-June 2015.

[5] J. Bater, G. Elliott, C. Eggen, S. Goel, A. Kho, and J. Duggan. SMCQL: Secure query processing for private data networks. *PVLDB*, 10(6):673–684, 2017.

[6] M. Benedikt, J. Leblay, and E. Tsamoura. Querying with access patterns and integrity constraints. *PVLDB*, 8(6):690–701, 2015.

[7] S. S. Chow, J.-H. Lee, and L. Subramanian. Two-party computation model for privacy-preserving queries over distributed databases. In *Proc. of NDSS*, San Diego, CA, USA, February 2009.

[8] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati. Fragmentation in presence of data dependencies. *IEEE TDSC*, 11(6):510–523, 2014.

[9] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati. An authorization model for multi-provider queries. *PVLDB*, 11(3):256–268, 2017.

[10] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati. An authorization model for query execution in the cloud. *VLDBJ*, 31(3):555–579, 2021.

[11] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati. Distributed query evaluation over encrypted data. In *Proc. of DBSec*, Calgary, Canada, July 2021.

[12] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Authorization enforcement in distributed query evaluation. *JCS*, 19(4):751–794, 2011.

[13] E. Dimitrova, P. Chrysanthis, and A. Lee. Authorization-aware optimization for multi-provider queries. In *Proc. of SAC*, Limassol, Cyprus, April 2019.

[14] N. Farnan, A. Lee, P. Chrysanthis, and T. Yu. PAQO: Preference-aware query optimization for decentralized database systems. In *Proc. of ICDE*, Chicago, IL, USA, March–April 2014.

[15] D. Gritzalis, G. Stergiopoulos, E. Vasilellis, and A. Anagnostopoulou. *Readiness Exercises: Are Risk Assessment Methodologies Ready for the Cloud?*, pages 109–128. Springer International Publishing, 2021.

[16] D. Gritzalis, M. Theocharidou, and G. Stergiopoulos. *Critical infrastructure security and resilience*. Springer, 2019.

[17] M. Guarnieri and D. Basin. Optimal security-aware query processing. *PVLDB*, 7(12):1307–1318, 2014.

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

[19] D. Kossmann. The state of the art in distributed query processing. *ACM CSUR*, 32(4):422–469, 2000.

[20] M. M. Kwakye and K. Barker. Privacy-preservation in the integration and querying of multidimensional data models. In *Proc of PST*, Auckland, New Zealand, December 2016.

[21] A. Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *JIIS*, 5(2):121–143, 1995.

[22] G. Li, J. Wu, J. Li, Z. Guan, and L. Guo. Fog computing-enabled secure demand response for internet of energy against collusion attacks using consensus and ACE. *IEEE Access*, 6:11278–11288, 2018.

[23] K. Y. Oktay, M. Kantarcioglu, and S. Mehrotra. Secure and efficient query processing over hybrid clouds. In *Proc. of ICDE*, San Diego, CA, USA, April 2017.

[24] R. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *Proc. of SOSP*, Cascais, Portugal, October 2011.

[25] C. Priebe, K. Vaswani, and M. Costa. EnclaveDB: A secure database using SGX. In *Proc. of S&P*, San Francisco, CA, USA, March 2018.

[26] A. Rheinländer, U. Leser, and G. Graefe. Optimization of complex dataflows with user-defined functions. *ACM CSUR*, 50(3):38:1–38:39, 2017.

[27] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proc. of SIGMOD*, Paris, France, June 2004.

[28] G. Salvaneschi, M. Köhler, D. Sokolowski, P. Haller, S. Erdweg, and M. Mezini. Language-integrated privacy-aware distributed queries. *Proc. ACM Program. Lang.*, 3(OOPSLA):1–30, 2019.

[29] S. Sharma, A. Burtsev, and S. Mehrotra. Advances in cryptography and secure hardware for data outsourcing. In *Proc. of ICDE*, Dallas, TX, USA, April 2020.

[30] C. Thoma, A. Lee, and A. Labrinidis. Behind enemy lines: Exploring trusted data stream processing on untrusted systems. In *Proc. of CODASPY*, Richardson, TX, USA, March 2019.

[31] S. Tu, M. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. *PVLDB*, 6(5):289–300, 2013.

[32] D. Vinayagamurthy, A. Gribov, and S. Gorbunov. StealthDB: A scalable encrypted database with full SQL query support. *PoPETS*, 2019(3):370–388, 2019.

[33] S. Xie, M. Mohammady, H. Wang, L. Wang, J. Vaidya, and Y. Hong. A generalized framework for preserving both privacy and utility in data outsourcing. *IEEE TKDE*, 2021 (early access).

[34] Q. Zeng, M. Zhao, P. Liu, P. Yadav, S. Calo, and J. Lobo. Enforcement of autonomous authorizations in collaborative distributed query evaluation. *IEEE TKDE*, 27(4):979–992, 2015.

Appendix A. Proofs of theorems

Theorem 4.1 (Monotonicity of the candidate set). *Let* $\mathtt{T(N)}$ *be a query tree plan and* $\mathtt{N^r}$ *the set of re-encryption nodes injected into* $\mathtt{T(N)}$ *to enable query evaluation.* $\forall n_x, n_y \in \mathtt{N} \cup \mathtt{N^r}$ *such that* n_x,n_y *are non-leaf nodes of* $\mathtt{T(N)}$*,* n_y *is a child of* n_x*, and* $n_y.A_p \subseteq R_x^{ip}$*:*

1. $\Lambda(n_x) \subseteq \Lambda(n_y)$*, if* n_x*,* $n_y \in \mathtt{N}$*;*
2. $\Lambda(n_z) \subseteq \Lambda(n_y)$*, if* $n_x \in \mathtt{N^r}$ *and* $n_z \in \mathtt{N}$ *is the parent of* n_x*.*

Proof: We separately prove the two conditions.

1. Assume, by contradiction, that $\exists S \in \Lambda(n_x)$ s.t. $S \notin \Lambda(n_y)$ (i.e., S is a candidate for the parent node n_x but not for the child node n_y). By Definition 4.1, S is not a candidate for n_y if it is not authorized for at least one among: *i)* the relation produced by its children n_l and n_r considering all visible attributes encrypted; *ii)* $n_y.A_p$ in plaintext; and *iii)* the relations produced by n_y considering all visible attributes encrypted. We separately prove that all these conditions contradict our hypothesis.

   i) If S is not authorized for the relation produced by n_l and n_r with all visible attributes encrypted (condition *i)*), but S is a candidate for n_x, then the profiles of n_l and/or n_r must include at least an (encrypted) attribute a that S cannot access or a pair of equivalent attributes a_i and a_j for which S does not have uniform visibility. We note however that no attribute is removed from the profile of R_y by the execution of the operation represented by n_y. Hence, a, a_i, and a_j also belong to the profile of the ancestors of n_l and n_r. To demonstrate that no attribute is removed from the profile of R_y, we analyze how the components of the profile are affected by operations. With respect to the (plaintext and encrypted) implicit component and the equivalence component, it is immediate to see from Figures 2 and 3 that no operation removes attributes from these components. With respect to the (plaintext and encrypted) visible component, on the other hand, while selection σ, cartesian product $\times$, and join $\bowtie$ operations do not remove attributes, projection π and group by γ operations remove attributes from the visible component of their operand. However, the attributes removed by these operations already belong to the implicit components of the profile (from which, as already observed, they never disappear). Indeed, since projections are pushed down in the tree, only attributes explicitly involved in operations in the query plan and those returned by the query survive projections operating on leaf nodes. Hence, the attributes removed by projections that do not operate on leaf nodes, or by group by operations are attributes on which some operation has been evaluated before the projections/group by. These operations include: *i)* selection σ, which however inserts the involved attribute(s) in either the implicit or the equivalent component of the result; *ii)* join $\bowtie$, which however inserts the involved attributes in the equivalent component of the result. Cartesian product ($\times$) does not explicitly operate on

any attribute, and attributes subject to aggregations either belong to the query result, or are involved in operations. Since no attribute can disappear from the profile of n_l and n_r, all the attributes in the visible components of the relation profiles are encrypted, and n_x is an ancestor of n_l and n_r, if S is not authorized for n_l or for n_r, S cannot be a candidate for n_x.

ii) If S is not authorized to access $n_y.A_p$ in plaintext (condition *ii)*), then S is also not a candidate for n_x since, by hypothesis, $n_y.A_p \subseteq R_x^{ip}$.

iii) If S is not authorized for the relations produced by n_y considering all visible attributes encrypted (condition *iii)*), then S cannot be a candidate for n_x (condition *1)* of Definition 4.1).

Therefore, $S \notin \Lambda(n_y) \Longrightarrow S \notin \Lambda(n_x)$, contradicting our hypothesis.

2. Since re-encryption operations do no have effect on the profile of its operand, the profile of R_x is the same as the profile of R_y. Hence, condition *1)* above applies between n_z and n_y. □

Theorem 4.2 (Existence of an authorized assignment). *Let* $\mathtt{T(N)}$ *be a query tree plan,* $\mathcal{S}$ *be a set of subjects, and* $\forall n \in \mathtt{N}$, $n.A_e$ *be the set of attributes that need to be re-encrypted for the evaluation of* n *and* $\Lambda(n)$ *be the set of candidates for* n*. If* $\forall n \in \mathtt{N}$, $\Lambda(n) \neq \emptyset$ *and,* $\forall a \in n.A_e$*, there exists at least a subject* $S \in \mathcal{S}$ *s.t.* $a \in \mathcal{P}_S$ *and* $R \subseteq \mathcal{P}_S \cup \mathcal{E}_S$*, with* R *the base relation to which* a *belongs, then there exists at least an extended query plan* $\mathtt{T'(N')}$ *of* $\mathtt{T(N)}$ *and an assignment* $\lambda : \mathtt{N'} \rightarrow \mathcal{S}$ *of subjects to nodes in* $\mathtt{T'(N')}$ *such that:*

1. $\lambda(n) = \lambda(n_p)$*, with* n_p *the parent of* n*, if* n *is a decryption operation;*
2. $\lambda(n) = \lambda(n_c)$*, with* n_c *the child of* n*, if* n *is an encryption operation;*
3. $\lambda(n) \in \Lambda(n)$*, otherwise;*

that does not violate any authorization.

Proof: We prove the existence of $\mathtt{T'(N')}$ and λ by construction. Given query plan $\mathtt{T(N)}$, we can assign each node $n \in \mathtt{N}$ to one of its candidate $\lambda(n) \in \Lambda(n)$, since by hypothesis $\Lambda(n) \neq \emptyset$. We then extend $\mathtt{T(N)}$ including the following three sets of nodes, and define the corresponding assignments.

- $\mathtt{N^e}$: set of encryption nodes. For each pair of nodes n and n_c in $\mathtt{T(N)}$ such that n is the parent of n_c in $\mathtt{T(N)}$ and $\exists \{a_i, \ldots, a_j\} \subseteq R_c^{vp}$ s.t. $\{a_i, \ldots, a_j\} \subseteq \mathcal{E}_{\lambda(n)}$, we insert an encryption node n_e for attributes $\{a_i, \ldots, a_j\}$ as child of n and parent of n_c. This makes $\lambda(n)$ authorized for n. Since $\{a_i, \ldots, a_j\} \subseteq R_c^{vp}$, $\lambda(n_c)$ is authorized for $\{a_i, \ldots, a_j\}$ in plaintext. Hence, setting $\lambda(n_e) = \lambda(n_c)$ does not violate any authorization.

- $\mathtt{N^d}$: set of decryption nodes. For each pair of nodes n and n_p in $\mathtt{T(N)}$ such that n is a child of n_p in $\mathtt{T(N)}$ and $\exists \{a_i, \ldots, a_j\} \subseteq R^{ve}$ s.t. $\{a_i, \ldots, a_j\} \subseteq n_p.A_p$, we insert a decryption node n_d for

attributes $\{a_i, \ldots, a_j\}$ as parent of n and child of n_p. Since $\{a_i, \ldots, a_j\} \subseteq n_p.A_p$, $\lambda(n_p)$ is authorized for $\{a_i, \ldots, a_j\}$ in plaintext. Hence, setting $\lambda(n_d)=\lambda(n_p)$ does not violate any authorization.

- $\mathtt{N}^{\mathtt{r}}$ set of re-encryption nodes. For each attribute a s.t. $\exists n \in \mathtt{N}$, $a \in n.A_e$, we insert a re-encryption node n_r for a as parent of the leaf node representing base relation R_i such that $a \in R_i^{vE}$. Since, by hypothesis, $\exists S \in \mathcal{S}$ such that $a \in \mathcal{P}_S$ and $R_i \subseteq \mathcal{P}_S \cup \mathcal{E}_S$, $\Lambda(n_r) \neq \emptyset$, and the choice of any $\lambda(n_r) \in \Lambda(n_r)$ does not violate any authorization.

We conclude that there exists an extended query plan $\mathtt{T}'(\mathtt{N}')$ and an assignment λ that does not violate authorizations. □

Theorem 5.1 (Complexity). *Let $\mathtt{T}(\mathtt{N})$ be a query tree plan, $\mathcal{A}$ be the set of attributes in the base relations of the plan, and $\mathcal{S}$ be the set of subjects. The complexity of the algorithm in Figure 6 is $O(|\mathtt{N}|\cdot|\mathcal{S}|\cdot|\mathcal{A}|)$ in time.*

PROOF: The algorithm in Figure 6 calls procedures: **Compute_Cost** (Figure 7), **Identify_Candidates** (Figure 8), **Compute_Assignment** (Figure 9), and **Extend_Plan** (Figure 10).

Procedure **Compute_Cost** has cost $O(|\mathtt{N}|\cdot|\mathcal{S}|)$, since it visits the tree $\mathtt{T}(\mathtt{N})$ (lines 1–2) and, for each node, computes the cost of evaluating the node at each of the subjects in $\mathcal{S}$ (**for each** loop at line 3).

Procedure **Identify_Candidates** has cost $O(|\mathtt{N}|\cdot|\mathcal{S}|)$, since it visits the tree $\mathtt{T}(\mathtt{N})$ (lines 1–2) and, for each node, in the worst case, it checks each subject in $\mathcal{S}$ to determine whether it is a candidate for the node (**for each** loop at line 20).

Procedure **Compute_Assignment** has cost $O(|\mathtt{N}|\cdot|\mathcal{S}|\cdot|\mathcal{A}|)$. Indeed, the procedure visits the tree $\mathtt{T}(\mathtt{N})$ (lines 44–45). For each non-leaf node (line 20), the procedure compares the costs of the candidates for the node. In the worst case, the set of candidates includes the whole set $\mathcal{S}$ of subjects (**for each** loop at line 19 in Figure 8). For each subject it then identifies and estimates the cost of the attributes that need to be decrypted, encrypted, and re-encrypted, respectively. We note that, in the worst case, the **for each** loops at lines 24, 26, and 29 consider all the attributes in $\mathcal{A}$ and that no attribute is considered by more than one of the **for each** loops. For each leaf node (line 3), the procedure instead checks which of the subjects in $\mathcal{S}$ can be a candidate for n by verifying its privileges over the attributes in the corresponding base relation R (**while** loop at line 8).

Procedure **Extend_Plan** has cost $O(|\mathtt{N}|)$ since it visits the tree $\mathtt{T}(\mathtt{N})$ (lines 1–2) and, for each node, possibly inserts an additional encryption/decryption node and recomputes the node profile. All these operations have constant cost.

The complexity of the algorithm in Figure 6 is then obtained by summing the costs of all the invoked procedures, that is, $O(|\mathtt{N}|\cdot|\mathcal{S}|)+O(|\mathtt{N}|\cdot|\mathcal{S}|)+O(|\mathtt{N}|\cdot|\mathcal{S}|\cdot|\mathcal{A}|)+O(|\mathtt{N}|) = O(|\mathtt{N}|\cdot|\mathcal{S}|\cdot|\mathcal{A}|)$. □

Theorem 5.2 (Correctness). *Let* $\mathtt{T(N)}$ *be a query tree plan and* $\mathcal{S}$ *be the set of subjects. If there exist an extended query tree plan* $\mathtt{T'(N')}$ *of* $\mathtt{T(N)}$ *and an assignment* $\lambda : \mathtt{N'} \rightarrow \mathcal{S}$ *such that:*

1. $\forall n \in \mathtt{N'}$*:* $\lambda(n) = \lambda(n_p)$*, with* n_p *the parent of* n*, if* n *is a decryption operation;* $\lambda(n) = \lambda(n_c)$*, with* n_c *the child of* n*, if* n *is an encryption operation; and* $\lambda(n) \in \Lambda(n)$*, otherwise;*
2. $\forall n \in \mathtt{N'}$*,* $\lambda(n)$ *is authorized for the profile of* n *and of its children (Definition 3.3);*

the algorithm in Figure 6 terminates and finds it.

PROOF: We first prove that the assignment produced by the algorithm satisfies the conditions of the theorem (*correctness*), and then show that, if such an assignment exists, the algorithm terminates (*termination*) and finds it (*completeness*).

- *Correctness.* We separately prove Condition 1 and Condition 2 of the theorem.

  1. The algorithm in Figure 6 calls procedure **Compute_Assignment** (Figure 9) to assign a subject to each node in the input query plan $\mathtt{T(N)}$. Procedure **Compute_Assignment** performs a visit of $\mathtt{T(N)}$ and, for each non-leaf node n, selects a subject S in $\Lambda(n)$ as the assignee of n (line 35). Indeed, $\Lambda(n)$ is populated by procedure **Identify_Candidates**, which inserts S into $\Lambda(n)$ only if S satisfies Definition 4.1 (lines 21-22, Figure 8). Hence, for each node $n \in \mathtt{N}$, $\lambda(n) \in \Lambda(n)$. Procedure **Compute_Assignment** also checks whether $\lambda(n)$ can re-encrypt: *i)* the attributes that need to be re-encrypted for the evaluation of n (line 41); and/or *ii)* the attributes that needed to be re-encrypted higher in the tree but whose re-encryption had been pushed down since the assignee of the ancestors of node n cannot perform it (line 36). Note that, to be considered for the re-encryption of attribute a, $\lambda(n)$ must have plaintext visibility over a (lines 36 and 41) and, being a candidate for n, $\lambda(n)$ is by construction a candidate for the re-encryption of a. If, when reaching a leaf in the tree, there exists at least an attribute a whose re-encryption has not been assigned, the procedure determines whether there is a subject S that can perform it and, if so, assigns the re-encryption to S (lines 5–15).
     Encryption and decryption nodes are injected by procedure **Extend_Plan** (Figure 10), which is executed after procedure **Compute_Assignment**. The assignee of each encryption node is the same as its child (line 18), and the assignee of each decryption node is the same as its parent (lines 9 and 13). Hence, the assignment computed by the algorithm in Figure 6 satisfies Condition 1 of the theorem.
  2. For each node $n \in \mathtt{N}$, by Condition 1 above, $\lambda(n) \in \Lambda(n)$ and is then authorized for the profile of n and of its children, assuming that all the visible attributes are encrypted. Given the assignment of operations to subjects computed by procedure **Compute_Assignment**, procedure

Extend_Plan inserts an encryption node between n and its parent n_p to encrypt all the attributes that n_p cannot access in plaintext. Note that decryption nodes inserted by procedure **Extend_Plan** do not violate authorizations since they involve only attributes in $n.A_p$ that, by Definition 4.1, any candidate subject in $\Lambda(n)$ can access in plaintext. Hence, the assignment computed by the algorithm in Figure 6 satisfies Condition 2 of the theorem.

- *Termination.* The algorithm in Figure 6 terminates since it invokes only procedures that terminate. Indeed, each of the procedures invoked by the algorithm in Figure 6 performs a visit of the query tree plan $\mathtt{T(N)}$ and, for each visited node, the **for**, **for each**, and **while** loops terminate, since they operate on finite sets (of attributes and of subjects).

- *Completeness.* We assume, by contradiction, that there exists an extended query tree plan $\mathtt{T'(N')}$ and an assignment λ that does not violate any authorization, but the algorithm in Figure 6 does not find it. The algorithm can fail in computing an assignment and the corresponding extended query tree plan due to two reasons: *i)* it does not find a subject S to which a node $n \in \mathtt{N}$ can be assigned without violating authorizations, or *ii)* it does not find a subject S that can re-encrypt an attribute a that requires re-encryption.
  We first note that, according to our assumption, $\Lambda(n) \neq \emptyset$, $\forall n \in \mathtt{N}$. Also, procedure **Identify_Candidates** (Figure 8) identifies all the subjects in $\Lambda(n)$. Indeed, for a node n, the procedure checks all the subjects in $\Lambda(n_l) \cup \Lambda(n_r)$ (line 18) and, according to Theorem 4.1, no other subject can be a candidate for n. We also note that, for leaf nodes, $\Lambda(n)$ is set to $\mathcal{S}$ (line 7). Therefore, for the parents of leaf nodes (as well as for nodes that do not satisfy Theorem 4.1) the procedure checks all the subjects (line 19). Procedure **Compute_Assignment** (Figure 9), which evaluates all the subjects in $\Lambda(n)$ to choose an assignment for n (line 20), will find an authorized assignment for n, if such an assignment exists. Note that the choice of any candidate subject in $\Lambda(n)$ can be made authorized by injecting encryption over the attributes that the subject cannot access in plaintext, as a child of n. As already discussed, this is always possible and is enforced by procedure **Extend_Plan** (Figure 10). We note that, according to our assumption, for each attribute a that needs to be re-encrypted, there exists a subject $S \in \mathcal{S}$ that is authorized to access attribute a in plaintext and all the attributes in the corresponding base relation in encrypted form. Since, in the worst case, procedure **Compute_Assignment** (Figure 9) evaluates each subject S in $\mathcal{S}$ and verifies whether S is authorized to access a in plaintext and the corresponding relation in either plaintext or encrypted form (lines 11-12), if a subject authorized for re-encryption exists, the procedure finds it. □