An Authorization Model for Multi-Provider Queries

Sabrina De Capitani di Vimercati
Università degli Studi di Milano
sabrina.decapitani@unimi.it

Sara Foresti
Università degli Studi di Milano
sara.foresti@unimi.it

Sushil Jajodia
George Mason University
jajodia@gmu.edu

Giovanni Livraga
Università degli Studi di Milano
giovanni.livraga@unimi.it

Stefano Paraboschi
Università di Bergamo
parabosc@unibg.it

Pierangela Samarati
Università degli Studi di Milano
pierangela.samarati@unimi.it

ABSTRACT

We present a novel approach for the specification and enforcement of authorizations that enables controlled data sharing for collaborative queries in the cloud. Data authorities can establish authorizations regulating access to their data distinguishing three visibility levels (no visibility, encrypted visibility, and plaintext visibility). Authorizations are enforced in the query execution by possibly restricting operation assignments to other parties and by adjusting visibility of data on-the-fly. Our approach enables users and data authorities to fully enjoy the benefits and economic savings of the competitive open cloud market, while maintaining control over data.

PVLDB Reference Format:
Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Giovanni Livraga, Stefano Paraboschi, and Pierangela Samarati. An Authorization Model for Multi-Provider Queries. *PVLDB*, XX(X): xxxx-yyyy, 201X.
DOI: https://doi.org/XXX

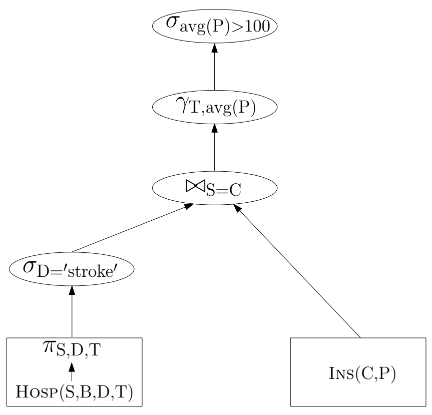
1. INTRODUCTION

Today's ICT scenarios are seeing an ever-growing explosion of data collection, sharing, and collaborative processing, as well as an ever-increasing need to efficiently perform extensive data analysis tasks over data produced and controlled by different parties (e.g., in medical or genomic data applications). The evolution of technology, and especially of the cloud computing paradigm, well responds to such demands and needs, offering a variety of storage and computation providers with different costs and performance guarantees. Multi-provider applications can leverage the richness and diversity of the cloud market by involving different parties depending on specific needs and economic convenience. Users and companies can then enjoy clear social and economic benefits in terms of convenient, scalable, and elastic availability of services. At the same time, however, data could be sensitive, proprietary, or simply subject to access restrictions that can affect the possibility of relying on external cloud providers for their management and processing [23]. Restricting processing within the premises of each individual data authority (i.e., the entity controlling the data) or at the user side, security concerns over data exposure can hinder the ability to fully benefit from the rich and diverse offer of the cloud market, and represent a significant barrier towards the evolution of the market and the related economic growth.

In this paper, we address this problem and propose a novel approach enabling collaborative and distributed query execution with the controlled involvement of cloud providers that might be not fully trusted to access the data content. Our goal is twofold: first, to allow data authorities to make their data available for possible collaborative processing, while maintaining control over them; second, to allow users accessing such data to leverage the rich and diverse offer of the cloud market, by relying on cloud providers for performing queries over such data.

The core of our proposal is a simple, yet flexible, authorization model that enjoys the great advantage of simplicity of specification and management. Each data authority can establish authorizations regulating the release to other subjects (i.e., users, providers, and other data authorities) of data under its control. Authorizations are specified by each authority independently (no cross-domain authorization or collaborative administration is required) and selectively grant visibility on the data to other subjects. Visibility can be granted either plaintext or encrypted. Subjects authorized for encrypted visibility over some data can perform computations (e.g., evaluate conditions or perform joins) over the data without accessing the actual data values. Leveraging the availability of solutions that support operations on encrypted data (e.g., CryptDB [19] and the SEEED framework over the SAP Hana DBMS [12]), this feature increases the spectrum of potential providers to which operations within a query can be assigned. Query execution can then selectively involve, in the different steps of the computation, different data authorities and cloud providers as deemed desirable for economic or performance reasons. Authorizations imposed by data authorities are enforced by applying encryption/decryption on-the-fly as needed to disable/enable data visibility as demanded by authorizations and operation requirements. Authorization enforcement will entail controlling not only direct data access, or release, but also accounting for information implicitly conveyed as a result of a computation.

Running example. For concreteness, but without loss of generality, we frame our work in the context of relational database systems. We consider queries of the general form "SELECT FROM WHERE GROUP BY HAVING" that can include joins among distinct relations under control of different data authorities. Execution of queries is performed according to a query plan established by the query optimizer. The query plan is represented as a tree `T(N)` whose leaves are base relations and whose non-leaf nodes are operations to be executed to perform the query. We assume the query plan to be produced with classical optimization criteria and, in particular, we assume that projections are pushed down to avoid retrieving data that are not of interest for the query. Graphically, we represent a leaf node as a square box that contains (the projection of) a source relation. We refer to leaf nodes as base relations. In this paper, we consider as a running example two data authorities: a hospital $\mathbb{H}$, storing relation HOSP(S,B,D,T), reporting SSN, Birth, Disease, and Treatment of hospitalized patients; and an insurance company $\mathbb{I}$ storing relation INS(C,P), reporting for each Customer the insurance Premium. We assume a user $\mathbb{U}$ and three cloud providers $\mathbb{X}$, $\mathbb{Y}$, $\mathbb{Z}$ offering computational power. Our running example considers the execution, on behalf of user $\mathbb{U}$, of query "SELECT T, avg(P) FROM HOSP JOIN INS ON S=C WHERE D='stroke' GROUP BY T HAVING avg(P)>100" retrieving, for each treatment given to patients hospitalized for stroke, the average insurance premium (if greater than USD100). Figure 1(a) illustrates a query plan for our query.


$\sigma_{\text{avg(P)}>100}$
$\gamma_{\text{T,avg(P)}}$
$\bowtie_{\text{S=C}}$
$\sigma_{\text{D='stroke'}}$
$\pi_{\text{S,D,T}}$
HOSP(S,B,D,T)
INS(C,P)

[SBDT,_]→$\mathbb{H}$
[C,P]→$\mathbb{H}$

[B,SDT]→$\mathbb{I}$
[CP,_]→$\mathbb{I}$

[SDT,_]→$\mathbb{U}$
[CP,_]→$\mathbb{U}$

[DT,S]→$\mathbb{X}$
[_,CP]→$\mathbb{X}$

[BDT,S]→$\mathbb{Y}$
[P,C]→$\mathbb{Y}$

[ST,D]→$\mathbb{Z}$
[C,P]→$\mathbb{Z}$

[DT,_]→any
[_,P]→any

(a) Query plan (b) Authorizations

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

Outline. The remainder of this paper is organized as follows. Section 2 presents our authorization model. Section 3 describes the concept of relation profile, capturing the informative content of a relation. Section 4 shows how protection requirements stated by authorizations must be considered to ensure that data are properly protected in query execution. Section 5 describes the use of on-the-fly encryption and decryption for protecting data in a computation, based on the assignment of query operations to subjects. Section 6 shows how to compute and distribute assignments. Section 7 presents the economic benefits of our solution. Section 8 discusses related work. Finally, Section 9 concludes the paper. The proofs of theorems can be found at http://spdp.di.unimi.it/dfjlps-vldb2018.pdf.

2. AUTHORIZATION MODEL

We assume a simple, yet expressive, authorization model in which each data authority specifies authorizations regulating the release of its data. Authorizations are defined at the fine-grained level of attribute specifying, for every attribute, whether a *subject* (i.e., a user, a data authority, or a provider) can have:

- *plaintext visibility*: the subject has complete visibility on the values of the attribute;
- *encrypted visibility*: the subject cannot view the plaintext values of the attribute, but can view an encrypted version of them;
- *no visibility*: the subject cannot view the values of the attribute at all (neither plaintext nor encrypted).

While plaintext and no visibility do not require explanation, since they correspond to traditional ways of regulating access, the encrypted visibility, which represents a characteristic and strength of our proposal, deserves some clarification. The reason behind the consideration of the encrypted visibility is to provide a subject with the ability to operate on an attribute for performing joins with other relations or for evaluating conditions on encrypted values (supported by the kind of encryption used), while not releasing to the subject the actual values of the attribute. In the authorization model we do not distinguish among different encryption schemes, so to leave the model simple and the approach flexible. In fact, expressing the encryption scheme in the authorizations would introduce considerable complexity in the specifications, without providing an actual advantage in the end. As the experience of null values shows (the introduction of multiple null values in SQL-92 was quickly deprecated), it is important to maintain specifications simple and intuitive. The distinction among encryption schemes will be made by the query optimizer in the generation of the query plan, depending also on the operations that are to be executed on the encrypted data (Section 6).

Consistently with standard security practice, we assume a "closed" policy for the specification of authorizations, meaning that only accesses explicitly authorized are allowed (i.e., 'no visibility' does not need to be specified, as it applies whenever the other two do not). Authorizations are then defined as follows.

DEFINITION 2.1 (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 view attributes P in plaintext and attributes E encrypted. Sets P and E are required to be disjoint. However, we note that an authorization that permits a subject S to access an attribute a in plaintext also allows S to access the encrypted version of the attribute. We assume that, for each relation, a subject can hold at most one authorization (the consideration of multiple authorizations would not increase expressivity). Since the set of subjects who might be involved in a query, and for whom release of data may be requested, may not be completely known a priori, a default authorization can be specified, which applies to all subjects for which no explicit authorization already exists for the interested relation. This is accommodated by the consideration of value 'any' as subject of the authorization.

We expect users to have authorizations that include plaintext attributes only, since users need to be able to access the queries' responses and manage keys for attributes encrypted in the computation. The data authority storing a relation can be expected to hold an authorization for accessing its content in plaintext (i.e., S storing $R(a_1,\ldots,a_n)$ is authorized for $[\{a_1,\ldots,a_n\},\_]\rightarrow S$). Providers and other data authorities may instead have authorizations that also include encrypted attributes, allowing them to operate on these attributes without disclosing their plaintext values. Figure 1(b) illustrates an example of authorizations for our running example. For simplicity, in the figure and in the remainder of this paper, we denote a set of attributes simply with the sequence of the attributes composing it, omitting the curly brackets and commas (e.g., SBDT stands for {S,B,D,T}).

3. RELATION CONTENT MODEL

To determine whether the release of a relation to a subject should be accepted according to authorizations, we introduce the concept of *relation profile* capturing the informative content of a base or derived (i.e., computed by a query) relation. In the following, we first illustrate how attributes that do not belong to a relation schema can influence the definition of its profile, and then formally define relation profiles.

3.1 Implicit and equivalent attributes

A relation resulting from a computation can convey information on attributes not explicitly appearing in its schema. This may happen due to the evaluation of a selection condition, of a grouping operation, or of a user defined function (udf) on attributes that are then removed from the relation schema through a projection. As a simple example, the relation resulting from "SELECT A FROM R WHERE B='10'", while containing only A in its schema, indirectly leaks information on the values of attribute B as well, and should therefore not be visible to subjects not authorized to see both A and B. A similar observation holds for the relation resulting from "SELECT A FROM R_1 JOIN R_2 ON A=B" which, while including only attribute A in its schema, conveys also information on B, as A and B satisfy the equality predicate (hence, granting visibility on A implies leaking also B). Capturing the informative content of a relation R (resulting from a computation) requires then to take into account such indirect information leakage and relationships among attributes, which we characterize through the concepts of *implicit* and *equivalent* attributes.

Implicit attributes. Implicit attributes are attributes not necessarily appearing in a relation schema but that have been taken into account in the computation of the relation. Basically, implicit attributes for a relation R are all those attributes that appear in a selection condition or grouping operation in the (sub-)query producing R. The information indirectly conveyed differs depending on the selection condition considered. For instance, a selection condition 'B=10' leaks the fact that all the tuples in the result have value of B equal to 10, disclosing B precisely even if it is not explicitly visible in the relation. A selection condition 'B>10' leaks instead the fact that the tuples appearing in the relation have a value for B greater than 10, but without leaking B's actual values. The evaluation of a GROUP BY clause over B is similar to the evaluation of equality condition 'B=*value*', where *value* may be unknown. Consistently with the fact that we operate at the schema level, we do not distinguish among the degrees of leakage and assume an attribute to be *implicitly visible* in a relation (i.e., indirectly exposed) if the attribute was taken into account – in some way – in the computation of the relation. The concept of implicit visibility applies to both plaintext and encrypted attributes.

Equivalent attributes. Equivalence among attributes captures the fact that some attributes have been connected in a computation (i.e., some conditions among them have been applied) and therefore visibility of one attribute indirectly leaks the other(s). Like for implicit attributes, the degree of such a leakage can depend on the condition enforced. For instance, condition 'A=B' implies precise leakage of the values of B from the visibility of A, while condition 'A>B' entails a partial leakage, as a subject viewing A can only infer the fact that B has a value lower than the one visible for A. Again, we do not consider different degrees of leakage (which would introduce considerable complexity and fuzziness in the approach, with limited advantages in the enforcement of authorizations), but simply capture such a connection between the attributes, considering them as equivalent from the point of view of authorization enforcement (as visibility of one entails visibility of the other). Given a relation R, we say that two attributes are *equivalent* if the (sub-)query producing R involves a condition comparing them. The equivalence relationship is symmetric and transitive. Different sets of equivalent attributes can exist for a given relation. The equivalence relationship can apply to both explicit as well as implicit attributes, and to plaintext as well as encrypted attributes.

3.2 Relation profile

We are now ready to define the profile of a relation, capturing the informative content carried by the relation in terms of attributes explicitly as well as implicitly visible and taking into account information conveyed by equivalent attributes. In the following, we refer to attributes explicitly visible in a relation as *visible* attributes, and to those implicitly leaked as *implicit*. In addition, attributes can be *plaintext* or *encrypted*.

DEFINITION 3.1 (RELATION PROFILE). *Let R be a relation. The* profile *of R is a 5-tuple of the form $[R^{vp}, R^{ve}, R^{ip}, R^{ie}, R^{\simeq}]$ where: R^{vp} and R^{ve} are the* visible *attributes appearing in R's schema in plaintext (R^{vp}) or encrypted (R^{ve}) form; R^{ip} and R^{ie} are the* implicit *attributes conveyed by R, in plaintext (R^{ip}) or encrypted (R^{ie}) form; and $R^{\simeq}$ is a disjoint-set data structure representing the closure of the equivalence relationship implied by attributes connected in R's computation.*

The profile of a base relation has all the elements but R^{vp} empty since it is assumed accessible in plaintext and does not carry any implicit content or equivalence relationship. (Note that plaintext accessibility of a relation does not imply that it is stored in plaintext but only that it is accessible in plaintext by the data authority.) Formally, the profile of a base relation $R(a_1,\ldots,a_n)$ is then $[\{a_1,\ldots,a_n\},\_,\_,\_,\_]$.

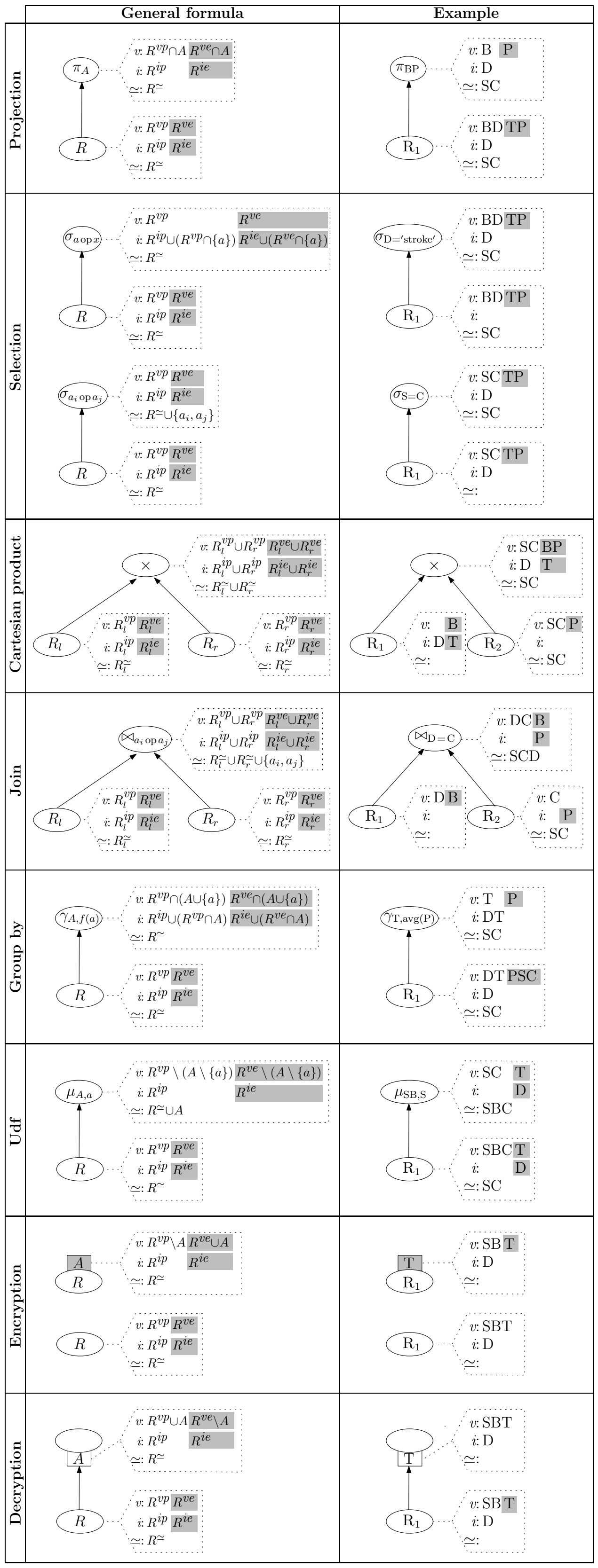


Figure 2: Graphical representation of the profiles resulting from relational, udf, and encryption/decryption operations

The profile of the relation resulting from a query depends on the profile of the operand relations and on the operators involved in its computation. Every operator only operates on visible attributes (i.e., attributes in R^{vp} and R^{ve}, which belong to the schema of the operand relation R), but it may affect also implicit attributes in the profile of the resulting relation. In the following, we illustrate the profile resulting from the application of projection, selection, cartesian product, join, group-by, and udf operators as well as encrypt/decrypt operators. In the treatment, with a slight abuse of notation, we will use symbol $\cup$ to denote the insertion of the equivalence relationship among a set A of attributes into $R^{\simeq}$. In other words, $R^{\simeq}\cup A$ adds A to $R^{\simeq}$ if no set in $R^{\simeq}$ intersects A, it merges all the sets intersecting A as well as A in a single set in $R^{\simeq}$, otherwise. $R_i^{\simeq}\cup R_j^{\simeq}$ implies inserting into $R_i^{\simeq}$ all the equivalence sets in $R_j^{\simeq}$ (or, equivalently, vice versa).

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: v (visible attributes R^{vp} and R^{ve}), i (implicit attributes R^{ip} and R^{ie}) and $\simeq$ (sets of equivalent attributes $R^{\simeq}$). Within visible and implicit attributes, we distinguish the encrypted ones (i.e., R^{ve} and R^{ie}) by representing them on a gray background. We represent an encryption operation as a gray box, containing the attributes to be encrypted, on top of the operand relation. We represent a decryption operation as a white box containing the attributes to be decrypted, below the node representing the operator for which the relation on which decryption works is operand. Figure 2 illustrates the graphical representation of the profiles resulting from the operations, reporting, for each operator, the general formula (on the left) and an example (on the right).

Projection (π). It returns a subset of the attributes in the schema of its operand. The profile of the resulting relation simply contains, in the visible attributes, only those attributes that have been projected. The implicit attributes and equivalence sets are the same as the ones of the operand.

Selection (σ). It returns a subset of the tuples of its operand, based on the evaluation of a condition on visible attributes. Since selection does not have any effect on the schema of the operand relation, the result has the same visible attributes as the operand. The other components of the profile depend on the kind of condition to be evaluated. For conditions of the form 'a op x', with x a value, attribute a is added to the implicit attributes (either encrypted or plaintext, consistently with how a is visible in the operand). For conditions of the form 'a_i op a_j', equivalence $\{a_i, a_j\}$ is added to the equivalence set. Note that attributes a_i and a_j must be either both plaintext visible or both encrypted visible for the evaluation of condition 'a_i op a_j'.

Cartesian product ($\times$). It returns the cartesian product of two operand relations R_l and R_r, that is, all possible combinations of their tuples. The plaintext/encrypted attributes visible or implicit in the resulting relation and the sets of equivalent attributes are then simply the union of the corresponding sets in the profiles of the operands.

Join ($\bowtie$). It returns a relation that contains the concatenation of the tuples of the operands R_l and R_r that satisfy a join condition C, which is a Boolean formula of basic conditions of the form 'a_i op a_j'. It is then equivalent to a selection operating on the cartesian product of the operands (i.e., $\sigma_C(R_l \times R_r)$). The profile of the result reflects then the information conveyed by both these operators. Also in this case, for each pair $\{a_i,a_j\}$ of attributes appearing together in a condition in C, a_i and a_j must be both plaintext or both encrypted for the evaluation of the join condition.

Group by (γ). It groups the operand relation by a given set of (plaintext or encrypted) attributes A, then evaluating an aggregate function f on an attribute a. For simplicity, we consider the attribute resulting from $f(a)$ with the same name as a.[1] The profile of the resulting relation contains, in the visible attributes, only those attributes on which the grouping (A) and aggregate function (a) operate (when $f(a)$ is COUNT($*$), only attributes in A are maintained). Attributes appearing in the grouping function (A) are added to the implicit attributes (to capture the possible information leakage from their grouping).

User defined function (μ). It performs a time-consuming procedural computation (e.g., machine learning and data analytics [6]) over the operand relation, elaborating the values of a set A of attributes (all plaintext or encrypted) in its schema. We assume a general udf operator with a set of attributes (A) as input and an attribute (a) as result. For simplicity, we assume the attribute in output to have the same name as one of the attributes in input.[1] The profile of the resulting relation has, as visible attributes, the attribute returned as output together with the visible attributes of the operand on which the udf does not operate. The implicit attributes are the same as the ones in the operand. The equivalence relationship is obtained from the one in the operand by adding the set of attributes to which the udf operates. This reflects the fact that the attribute in output depends on all the attributes on which the udf has operated.

Encryption. It changes a relation by encrypting some of its plaintext attributes. The result has the same profile as the operand, apart from the fact that the attributes on which encryption is applied are moved from visible plaintext to visible encrypted.

Decryption. It changes a relation by decrypting some of its encrypted attributes. The result has the same profile as the operand, apart from the fact that the attributes on which decryption is applied are moved from visible encrypted to visible plaintext.

Figure 3 illustrates the profiles associated with the relations resulting from the different operations of our running example. Each node has, on its left, the user and a set of providers (we will elaborate on this in the next section). Also, note that there are no encryption/decryption operations, as they do not appear in the original query plan; we will illustrate how and why the query plan is extended with them in Section 5. In the following, given a query plan, we use the term node to denote one of its components (base relation or an operation) and the term relation to denote either a base relation or the result of an operation (represented by an internal node). Given a node n_x, representing an operation, R_x denotes the relation resulting from it.

As stated, profiles allow us to capture the informative content of a relation resulting from a computation (which

[1]This simplification does not introduce limitations, since our model can be easily extended to support renaming.

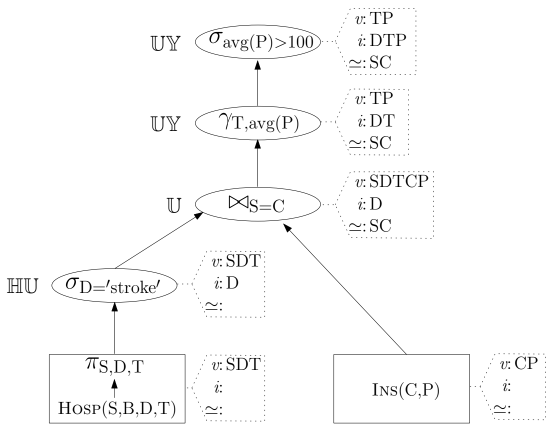


Figure 3: Query plan with profiles and authorized assignees

depends on the different computation steps). The following theorem proves that in a query plan: *i)* attributes appearing in the profile of the relation resulting from an operation will survive in the profiles of relations resulting from subsequent operations (i.e., they can move from one component to another, but they cannot be removed from the profile), and *ii)* equivalence sets can only increase going up in the query plan (i.e., when an attribute is inserted into an equivalence set, it cannot be removed from it).

THEOREM 3.1. *Let* T(N) *be a query plan.* $\forall n_x, n_y \in$ N *s.t.* n_y *is a descendant of* n_x*:*

i) $(R_y^{vp} \cup R_y^{ve} \cup R_y^{ip} \cup R_y^{ie} \cup \{A \mid A \in R_y^{\simeq}\}) \subseteq (R_x^{vp} \cup R_x^{ve} \cup R_x^{ip} \cup R_x^{ie} \cup \{A \mid A \in R_x^{\simeq}\})$

ii) $\forall A \in R_y^{\simeq} : \exists A' \in R_x^{\simeq}, A \subseteq A'$.

4. AUTHORIZED VISIBILITY AND ASSIGNMENT

The definition of relation profile (Definition 3.1) allows us to capture the informative content carried by a relation, and therefore to regulate query execution ensuring obedience to authorizations. Such regulations concern both visibility of relations as well as execution of operations in the query plan. Since a computation might involve different base relations, different authorization sets (and authorities) might be involved in the control for the release of a derived relation. We will elaborate on this in Section 6. In this section, for simplicity, we assume an overall view of the authorizations and we use notation $\mathcal{P}_S$ ($\mathcal{E}_S$, resp.) as a short-hand for the abstract concept summarizing the attributes that subject S is authorized to access in plaintext (encrypted, resp.) form. In other words, $\mathcal{P}_S = \{a \in P \mid [P,E] \rightarrow S\}$ and $\mathcal{E}_S = \{a \in E \mid [P,E] \rightarrow S\}$. Figure 4 shows the authorizations for our running example and the corresponding overall views for the different subjects.

The following definition captures the authorization control on a relation (based on its profile) to determine whether releasing it to a subject obeys authorizations, taking into account also information leakage caused by implicit attributes and equivalence relationships.

DEFINITION 4.1 (AUTHORIZED RELATION). *Let R be a relation with profile $[R^{vp}, R^{ve}, R^{ip}, R^{ie}, R^{\simeq}]$. A subject S is* authorized *for R iff:*

	Authorizations		**Authorized attributes**	
Subject	HOSP(S,B,D,T)	INS(C,P)	Plaintext	Encrypted
$\mathbb{H}$	$[\mathrm{SBDT},\_]\rightarrow\mathbb{H}$	$[\mathrm{C},\mathrm{P}]\rightarrow\mathbb{H}$	$\mathcal{P}_{\mathbb{H}}$=SBDTC	$\mathcal{E}_{\mathbb{H}}$=P
$\mathbb{I}$	$[\mathrm{B},\mathrm{SDT}]\rightarrow\mathbb{I}$	$[\mathrm{CP},\_]\rightarrow\mathbb{I}$	$\mathcal{P}_{\mathbb{I}}$=BCP	$\mathcal{E}_{\mathbb{I}}$=SDT
$\mathbb{U}$	$[\mathrm{SDT},\_]\rightarrow\mathbb{U}$	$[\mathrm{CP},\_]\rightarrow\mathbb{U}$	$\mathcal{P}_{\mathbb{U}}$=SDTCP	$\mathcal{E}_{\mathbb{U}}=\_$
$\mathbb{X}$	$[\mathrm{DT},\mathrm{S}]\rightarrow\mathbb{X}$	$[\_,\mathrm{CP}]\rightarrow\mathbb{X}$	$\mathcal{P}_{\mathbb{X}}$=DT	$\mathcal{E}_{\mathbb{X}}$=SCP
$\mathbb{Y}$	$[\mathrm{BDT},\mathrm{S}]\rightarrow\mathbb{Y}$	$[\mathrm{P},\mathrm{C}]\rightarrow\mathbb{Y}$	$\mathcal{P}_{\mathbb{Y}}$=BDTP	$\mathcal{E}_{\mathbb{Y}}$=SC
$\mathbb{Z}$	$[\mathrm{ST},\mathrm{D}]\rightarrow\mathbb{Z}$	$[\mathrm{C},\mathrm{P}]\rightarrow\mathbb{Z}$	$\mathcal{P}_{\mathbb{Z}}$= STC	$\mathcal{E}_{\mathbb{Z}}$=DP
any	$[\mathrm{DT},_]\rightarrow$any	$[_,\mathrm{P}]\rightarrow$any	$\mathcal{P}_{\text{any}}$=DT	$\mathcal{E}_{\text{any}}$=P

Figure 4: Authorizations and corresponding overall views for the subjects of our running example

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

According to Definition 4.1, a subject S is authorized to access a relation R iff the following three conditions hold: *1)* S is authorized to access in plaintext all the (visible or implicit) attributes represented in plaintext in R; *2)* S is authorized to access in plaintext or in encrypted form all the (visible or implicit) attributes represented in encrypted form in R; *3)* S is authorized to access in the same form (either plaintext or encrypted) all the equivalent attributes, that is, attributes that appear together in an equivalence set in $R^{\simeq}$ (uniform visibility).

Conditions 1 and 2 correspond to a simple enforcement of authorizations, taking into account both the visible and implicit attributes. Also, condition 2 considers the fact that subjects authorized for plaintext visibility over an attribute can also have encrypted visibility over the same (since the encrypted representation conveys less information than the plaintext one). Condition 3 enforces control on indirect information leakage caused by equivalence relationships established in query computation, to prevent unauthorized exposure of information. It requires the subject to have the authorizations for the attributes in equivalence sets, since the relation implicitly carries information about them. In other words, since they leave a trace in the computation result, all attributes in equivalence sets are always treated as implicit attributes. It also imposes that, within each equivalence set, the authorizations be the same (either plaintext or encrypted) for all attributes in the set. In fact, equivalence relationships in a profile express the fact that some attributes have been related in a computation (e.g., an equi-join operation) and therefore visibility of one attribute in an equivalence set leaks information on the other attributes in the same set. Imposing uniform visibility allows us to account for such inference channels, blocking them when not consistent with the authorizations. Note that uniform visibility must be satisfied for all attributes in an equivalence set, regardless of whether they actually belong to the relational schema (i.e., they are visible).

EXAMPLE 4.1. *Consider the authorizations in Figure 4 and a relation R with profile* $[\mathrm{P}, \mathrm{BSC}, \_, \_, \{\mathrm{SC}\}]$:

- $\mathbb{Y}$ *is authorized for* R;
- $\mathbb{H}$ *is not authorized for* R (*condition 1, attribute* P);
- $\mathbb{U}$ *is not authorized for* R (*condition 2, attribute* B);
- $\mathbb{I}$ *is not authorized for* R (*condition 3, attributes* SC).

Note that the enforcement of uniform visibility entails a possibly counter-intuitive result: a subject could be not authorized for a relation due to its plaintext visibility over some attributes, while another subject that, on these attributes, has only encrypted visibility could be authorized. For instance, with reference to Example 4.1, $\mathbb{I}$ is not authorized for R because it has plaintext visibility over C and encrypted visibility over S (and the equivalence between C and S could leak S to $\mathbb{I}$), while $\mathbb{Y}$ is authorized for R since it has only encrypted visibility over C and S, and therefore cannot draw any inference from R.

Definition 4.1 regulates if a subject can be authorized for a relation, based on its authorizations and on the relation profile. Another aspect involved in the enforcement of authorizations in our context concerns regulating the assignment of operations within a query plan to authorized subjects. An operation of the query plan, corresponding to a non-leaf node in the tree, operates on one or two operand relations, and produces a relation as output. A subject can be considered authorized for the execution of an operation if and only if it is authorized for all the relations involved: the operand(s) as well as the result. The authorized visibility for the operand(s) is needed since otherwise the subject could not access them. The authorized visibility for the result enforces the control over the information entailed by the execution of the operation itself. This is captured by the following definition.

DEFINITION 4.2 (AUTHORIZED ASSIGNEE). *Let* $\mathtt{T(N)}$ *be a query plan,* $n\in\mathtt{N}$ *be a non-leaf node,* n_l *and* $n_r\in\mathtt{N}$ *be its children (if any) producing relations* R_l *and* R_r*, and* $\mathcal{S}$ *be a set of subjects. Subject* $S\in\mathcal{S}$ *is an* authorized assignee *of* n *over* R_l *and* R_r *iff* S *is authorized for* R_l*, for* R_r*, and for the relation produced by* n*, according to Def. 4.1. Function* $\lambda : \mathtt{N} \rightarrow \mathcal{S}$ *is said to be an* authorized assignment function *for* $\mathtt{T(N)}$ *iff* $\forall n \in \mathtt{N}$, $\lambda(n)$ *is an authorized assignee of* n.

Subjects appearing on the left-hand side of each node in Figure 3 are authorized assignees for the node. Leaf nodes do not have any assignee since they remain with the data authority holding the interested base relation.

5. EXTENDED PLANS AND ENCRYPTION/DECRYPTION

Given a query plan, our goal is to produce an authorized assignment of operations to subjects. While the definitions in Section 4 accounted for the possible presence of encrypted attributes, the original query plan, including only operations requested by the query computation, does not include any encryption/decryption operation. Encryption and decryption operations are inserted on-the-fly by our approach to adjust visibility of attributes as required by operation requirements or authorizations. Encryption protects attributes so to permit the assignment of operations to subjects that could not be considered otherwise. Decryption permits accessing plaintext values of encrypted attributes when needed in the computation. For instance, assume that, for the query plan in Figure 3, all operations but the final selection ($\sigma_{\text{avg(P)}>100}$) could be performed on encrypted values. If all attributes were encrypted at their source (and avg(P) decrypted only for the last operation), more subjects could be considered for executing operations in the query. Figure 5 illustrates the query plan in Figure 3 extended to consider such encryption and decryption operations, reporting on the left-hand side of each node the subjects that could now be considered for the execution of the

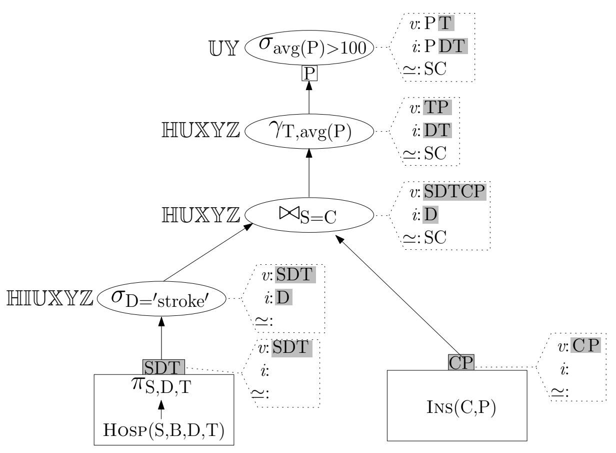


Figure 5: An extended query plan

node's operation. The specific encryption scheme to apply for the encryption of different attributes is decided by the query optimizer in the analysis of the query plan, depending on the kind of operations to be supported over such attributes. For instance, deterministic symmetric encryption can be used to efficiently and securely support evaluation of equality conditions in joins and selections.

A query plan T′ that is obtained by inserting encryption and decryption operations into another query plan T is called an *extended query plan* for T and is defined as follows.

Definition 5.1 (Extended Query Plan). *Let* T(N) *be a query plan. A query plan* T′(N) *is an* extended query plan *for* T *iff* T′ *is* T *enriched with some encryption and decryption operations.*

As said, encrypting attributes enables the consideration, for the assignment of an operation, of subjects not otherwise authorized for the execution of the operation. However, the encryption needed to make assignments authorized eventually depends on the actual subjects to which operations are assigned (e.g., P would need to be encrypted for assigning the execution of the join to $\mathbb{X}$ but could remain in plaintext if the join is assigned to $\mathbb{Y}$). There are basically two opposite approaches that can be followed in the insertion of encryption/decryption operations in the query plan, corresponding to maximizing or minimizing visibility of attributes. Maximizing visibility corresponds to always leave visibility of data in the clear, applying encryption only when strictly needed for protecting attributes visibility from the subject executing a specific operation. Minimizing visibility corresponds to always apply encryption by default, decrypting attributes only as needed for operation execution. Each of the two extremes has some pros and cons. Maximizing visibility by default can avoid unnecessary encryption/decryption operations and allows for operating as much as possible on plaintext data, but could reduce the number of subjects to which an operation can be assigned. For instance, suppose that attribute D is not encrypted for the execution of the selection operation ($\sigma_{\text{D='stroke'}}$), since such an operation is assigned to $\mathbb{H}$, which can see D in plaintext. Then, provider $\mathbb{Z}$ cannot be considered for the join since it does not have the authorization for plaintext visibility of D. In fact, encrypting D only for the join would not protect it from the possible leakage caused by the prior evaluation of the condition (as a matter of fact, D would remain in the implicit plaintext component of the profile of all relations computed after the selection over plaintext attribute D). Maximizing visibility of attributes at a given step may then rule out the consideration of possible subjects in subsequent steps of the query plan. Minimizing visibility, while not affecting the choice of subjects for subsequent operations in the query plan, could result in more encryption/decryption operations than needed. For instance, encrypting D before the execution of the selection operation may eventually result unnecessary, if $\mathbb{Z}$ were not the best choice for the join anyway, implying an overhead for query execution (encryption and possible less efficient evaluation of the condition) which could have been avoided.

To avoid predetermining one of the possible scenarios above, we adopt a more flexible approach by first determining the candidate subjects for operations, and then injecting encryption and decryption operations only as needed, depending on the decided assignment of operations to subjects. The query optimizer can then decide assignments of operations based on costs and performance aspects. Of course, assignment of operations to subjects must be bounded by the authorizations and the operation requirements, which can limit the application of encryption (as some operations need to access some attributes in plaintext for execution). As for authorizations, for example, while it is desirable for the execution of the join operation to possibly consider $\mathbb{X}$ (since S and C could be encrypted for that), it does not make sense to consider $\mathbb{I}$ since, as already noted, its non-uniform visibility over S and C (it is authorized to view C in plaintext but S only in encrypted form) rules it out from consideration (condition 3 of Definition 4.1). With respect to operation requirements, an attribute should not be encrypted if the operation to be executed on it requires accessing the attribute's plaintext values. For instance, if the encryption scheme available for P does not support range conditions, the possibility of encrypting avg(P) for assigning the last selection operation should be excluded. For operations that are not supported by cryptographic techniques (not existing or not available to the application), we assume the optimizer to specify the need for maintaining data in plaintext for execution of the operation. For each node we then have a set A_p of attributes that are needed in plaintext.

To define the potential candidates for an operation, we then first need to characterize the operation requirements, which may limit the application of encryption. We capture this by defining the *minimal visibility* needed over an operand to allow the evaluation of an operator. For instance, in our running example, we assume that the execution of the last selection in the query plan needs to view avg(P) in plaintext, while all other attributes can be encrypted. Intuitively, the minimum required view over an operand for the execution of an operation is the operand relation where all the (visible) attributes, but those that need to be in plaintext for operation execution, are encrypted, as formally captured by the following definition.

Definition 5.2 (Minimum Required View). *Let* T(N) *be a query plan,* $n \in$N *be a non-leaf node,* n_y *be one of its children, producing relation* R_y*, and* A_p *be the set of attributes of* R_y *that must be in plaintext for the execution of* n*. The* minimum required view *over* n_y *for the execution of* n *is relation* $\bar{R}_y$=$\texttt{decrypt}(A_p,\texttt{encrypt}(R_y^{vp} \setminus A_p, R_y))$.

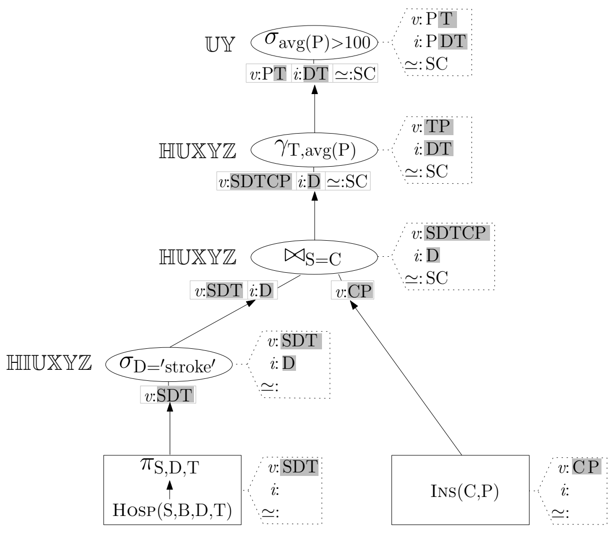


Figure 6: Minimum required views and assignment candidates

In the definition above and in the following, $\texttt{encrypt}(A,R)$ denotes the encryption of attributes A in R and $\texttt{decrypt}(A,R)$ denotes the decryption of attributes A in R. Figure 6 illustrates (in dotted boxes on the arcs from the operands to the operations) the profiles of the minimum required views for our running example. The profiles associated with nodes are those that result assuming as operands such minimum required views. For instance, the minimum required view over Ins for the execution of the join has all attributes (CP) visible and encrypted.

Minimum required views allow us to take into account the visibility requirements for operation execution: only subjects authorized for the minimum required views can be *candidates for the assignment* (since for them the operands could be protected with encryption without affecting operation execution). This is captured by the following definition.

Definition 5.3 (Assignment Candidates). *Let* $\texttt{T(N)}$ *be a query plan,* $n{\in}\texttt{N}$ *be a non-leaf node,* $n_l,n_r{\in}\texttt{N}$ *be its children (if any), and* $\mathcal{S}$ *be a set of subjects. A subject* $S{\in}\mathcal{S}$ *is a* candidate *for the execution of* n *iff* S *is an authorized assignee of* n *over* $\bar{R}_l$ *and* $\bar{R}_r$ *according to Def. 4.2.* Candidate assignment function $\Lambda : \texttt{N} \rightarrow 2^{\mathcal{S}}$ *associates with each* $n \in \texttt{N}$ *the set of candidates for the execution of* n*.*

Figure 6 illustrates assignment candidates for the operations of our running example.

The set of candidates along a query plan enjoys a nice monotonic behavior. The set of candidates of n's ancestors is a subset of the set of n's candidates. This applies to any node representing an operation that does not need to operate on plaintext attributes or that, doing so, leaves an implicit trace of such attributes (i.e., causes them to be included in the implicit attributes of the result's profile). In fact, all such attributes will also remain implicit plaintext in the profile of the minimum required view of any node n_x ancestor of n, and therefore, by definition, any candidate for n_x is certainly also a candidate for n. This is formalized by the following theorem.

Theorem 5.1. *Let* $\texttt{T(N)}$ *be a query plan,* $n{\in}\texttt{N}$ *be a non-leaf node* $n_l,n_r{\in}\texttt{N}$ *be its non-leaf children, if any.* $\bar{R}_l^{vp}{\cup}\bar{R}_r^{vp} \subseteq \bar{R}^{ip} \Longrightarrow \Lambda(n_x) \subseteq \Lambda(n), \forall n_x$ *ancestor of* n*.*

This monotonic behavior can be easily observed in Figure 6, where the set of candidates for each node decreases going up in the query plan.

The set of candidates for a node are *all and only* those subjects that can be made authorized assignees (Definition 4.2), assuming to extend the query plan with encryption/decryption operations, as stated by the following theorem.

Theorem 5.2. *Let* $\texttt{T(N)}$ *be a query plan, and* Λ *be a candidate assignment function for it:*

i) $\forall \texttt{T}', \lambda$, *and* $n \in \texttt{N}$, *if* $\texttt{T}'$ *is an extended query plan for* $\texttt{T}$ *and* λ *is an authorized assignment for* $\texttt{T}'$, *then* $\lambda(n) \in \Lambda(n)$.
ii) $\forall \lambda$, *if* $\forall n \in \texttt{N}, \lambda(n) \in \Lambda(n)$, *then there exists an extended query plan* $\texttt{T}'$ *for* $\texttt{T}$ *such that* λ *is an authorized assignment for* $\texttt{T}'$.

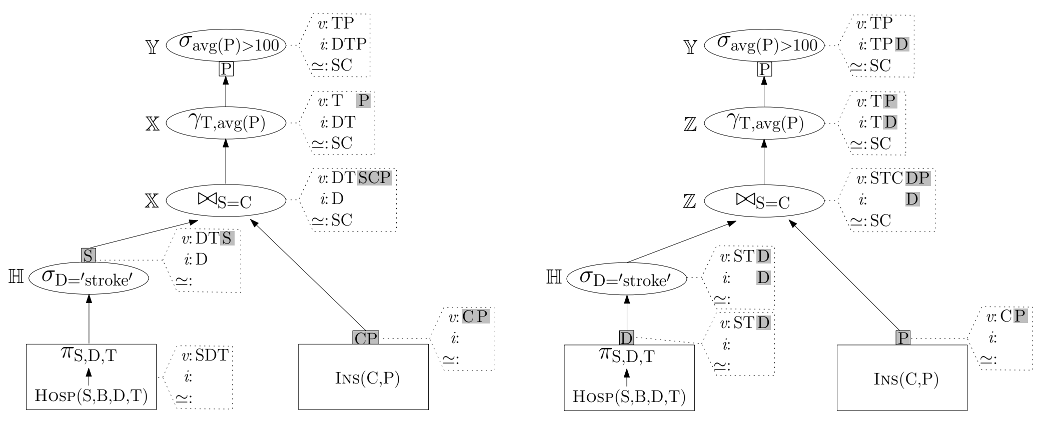
In other words: *i)* any assignment that can be made authorized by inserting some encryption and decryption operations is included in Λ, and *ii)* any assignment included in Λ can be made authorized by inserting some encryption and decryption operations.

Given a query plan $\texttt{T}$ and a possible assignment λ of operations taken from the potential candidates Λ, there are different ways in which encryption and decryption could be inserted to make λ authorized. For instance, enforcing all encryptions corresponding to the minimum required views (as in Figure 6) could work. However, it is desirable to aim to avoid the use of encryption if not needed for protection and therefore to produce a plan that encrypts only those attributes that need to be encrypted for obeying authorizations (and later decrypts them if needed for the execution of an operation). This is captured by the following definition.

Definition 5.4 (Minimally Extended Query Plan). *Let* $\texttt{T(N)}$ *be a query plan,* Λ *be a candidate assignment function for it, and* λ *be a function* $\lambda : \texttt{N} \rightarrow \mathcal{S}$ *such that,* $\forall n \in \texttt{N} : \lambda(n) \in \Lambda(n)$. *A* minimally extended authorized query plan *for* $\texttt{T}$ *is an extended query plan* $\texttt{T}'$ *obtained by possibly complementing each node* $n \in \texttt{N}$ *with decryption and encryption operations to precede and follow, respectively,* n*'s operator execution as follows:*

i) $\texttt{decrypt}(A_p \cap R_l^{ve}, R_l)$, $\texttt{decrypt}(A_p \cap R_r^{ve}, R_r)$ *with* A_p *the set of attributes needed in plaintext for the execution of* n, *and* R_l *and* R_r *the operands of* n;
ii) $\texttt{encrypt}((\mathcal{E}_{S_o} \cap R^{vp}) \cup A,\ R)$, *with* n_o *the parent of* n *and* $S_o = \lambda(n_o)$ *its assignee, and* $A = (R_o^{ip} \cap R^{vp}) \cap (\bigcup_x \mathcal{E}_{S_x})$ *with* n_x *an ancestor of* n *and* $\lambda(n_x) = S_x$ *its assignee.*

A minimally extended authorized query plan: *i)* decrypts attributes for an operation n when they appear encrypted in an operand but need to be accessed in plaintext for executing the operation; *ii)* encrypts attributes after an operation n when the next operation n_o (to which n is operand) either is to be executed by a subject that cannot access these attributes in plaintext or causes these attributes to be added to the set of implicit attributes (R_o^{ip}) and some subjects



$\mathcal{P}_{\mathbb{H}}$:SBDTC
$\mathcal{E}_{\mathbb{H}}$:P

$\mathcal{P}_{\mathbb{I}}$:BCP
$\mathcal{E}_{\mathbb{I}}$:SDT

$\mathcal{P}_{\mathbb{U}}$:SDTCP
$\mathcal{E}_{\mathbb{U}}$:-

$\mathcal{P}_{\mathbb{X}}$:DT
$\mathcal{E}_{\mathbb{X}}$:SCP

$\mathcal{P}_{\mathbb{Y}}$:BDTP
$\mathcal{E}_{\mathbb{Y}}$:SC

$\mathcal{P}_{\mathbb{Z}}$:STC
$\mathcal{E}_{\mathbb{Z}}$:DP

$\mathcal{P}_{\text{any}}$:DT
$\mathcal{E}_{\text{any}}$:P

(a) A minimally extended query plan (b) A minimally extended query plan (c) Authorized attributes

Figure 7: Minimally extended authorized query plans and authorized attributes for the plan in Figure 1

to which subsequent operations are assigned cannot access them in plaintext. The reason for encrypting attributes that are added to the implicit component by the next operation is that not doing so would cause indirect leakage of plaintext information on the attributes, which would invalidate assignment of subsequent operations.

Figures 7(a-b) illustrate two minimally extended authorized query plans for our running example assuming operations allocated to the subject indicated on the left-hand side of each node. For convenience of the reader, sets $\mathcal{P}$ and $\mathcal{E}$ of each subject (copied from Figure 4) are repeated in Figure 7(c). In the plan in Figure 7(a): S, C, and P are encrypted before being passed to $\mathbb{X}$, since $\mathbb{X}$ cannot access them in plaintext. In the plan in Figure 7(b), P is encrypted before being passed to $\mathbb{Z}$, since $\mathbb{Z}$ cannot access it in plaintext, while D is encrypted before executing the selection (i.e., the condition on D will have to be dispatched formulated on encrypted values) so not to leave an implicit plaintext trace in the computation given that $\mathbb{Z}$, executing subsequent steps, cannot access D in plaintext.[2] In both plans, avg(P) is decrypted before the execution of the final selection that needs to access plaintext values. Encryption and decryption operations are assigned to the same subjects as the nodes they complement. Indeed, the subject authorized for a node is also clearly authorized for the preceding decryption (of attributes that are needed in plaintext for the operation) and for the following encryption (of attributes available in plaintext).

A minimally extended query plan: *i)* makes the given assignment authorized, and *ii)* does so by encrypting a minimal set of attributes, as stated by the following theorem.

Theorem 5.3. *Let* T(N) *be a query plan,* λ *be an assignment function s.t.* $\forall n \in$ N, $\lambda(n) \in \Lambda(n)$, *and* T′ *be a minimally extended authorized query plan for* T *according to Def. 5.4:*

i) λ *is an authorized assignment for* T′ *(Def. 4.2);*
ii) *any other extended query plan of* T *(Def. 5.1) for which* λ *is an authorized assignment involves in encryption operations a superset of the attributes encrypted in* T′.

[2]Note that this does not necessarily imply the evaluation of the condition in encrypted form. Since $\mathbb{H}$ is the authority over D and it knows the encryption key (it encrypts D itself), $\mathbb{H}$ can operate on plaintext values and encrypt D afterwards.

6. COMPUTING AND DISTRIBUTING ASSIGNMENTS

The results of the previous section prove that, for any operation in the query plan, only subjects in the operation's candidate set need to be considered (Theorem 5.2, *i*). Also, any of them would do since any assignment taken from the candidate set can be made authorized by inserting proper encryption and decryption operations (Theorem 5.2, *ii*). This means that the query optimizer can work with classical cost and performance considerations and choose for the execution of an operation the subject, in the operation's candidate set, that is considered preferable. Note that determining the set of candidates for each operation does not require evaluating all subjects and their authorizations. In fact, the candidates of a node may naturally limit the candidates of its ancestors (Theorem 5.1).

Query operation assignment entails, besides assigning operations to candidates, also establishing and distributing keys for attributes that need to be encrypted/decrypted in the query plan execution. The only constraint on key establishment is that attributes involved in some conditions comparing them in encrypted form need to be encrypted with the same key. To ensure this, we simply require attributes appearing together in an equivalence set to be encrypted with the same key (even if they are encrypted after they have been compared, using the same key would not provide any leakage as they are indeed equivalent). As per Theorem 3.1, it is sufficient to look at the equivalence sets in the profile of the root to determine which attributes should be encrypted with the same key. We then define the keys to be established for a query plan execution as follows.

Definition 6.1 (Query Plan Keys). *Let* T(N) *be a minimally extended authorized query plan,* n_T *be its root, and* A_k *be the set of attributes involved in encryption operations. Let* $\mathcal{A} = \{\{A_k \cap A_j\} \mid A_j \in R_T^{\simeq}\} \cup \{\{a\} \mid a \in A_k, \nexists A_j \in R_T^{\simeq}, a \in A_j\}$. *The set* $\mathcal{K}_T$ *of keys for* T *is* $\mathcal{K}_T = \{k_A | A \in \mathcal{A}\}$, *with* k_A *an encryption key.*

In the definition, the family of sets $\mathcal{A}$ clusters attributes to be encrypted based on the equivalence sets in the root profile (attributes appearing together in an equivalence set belong to the same set in $\mathcal{A}$, while attributes not belonging to any equivalence set appear as singletons). The key associated with an attribute (or set thereof) will be distributed only to the subjects in charge for its (their) encryption and (possible) decryption. Since such subjects are authorized for the encryption/decryption operation (i.e., they are authorized for plaintext visibility of the attributes to be encrypted/decrypted in the operand relation), key distribution obeys authorizations. For instance, for the query plan in Figure 7(a), $\mathcal{A} = \{\mathrm{SC,P}\}$, resulting in k_{SC} distributed to $\mathbb{H}$ and $\mathbb{I}$, and k_{P} distributed to $\mathbb{I}$ and $\mathbb{Y}$. For the query plan in Figure 7(b), $\mathcal{A} = \{\mathrm{D,P}\}$, resulting in k_{D} distributed to $\mathbb{H}$, and k_{P} distributed to $\mathbb{I}$ and $\mathbb{Y}$.

Query operation assignments can then be performed by the query optimizer, as follows:

1. perform a post-order visit of the query plan identifying candidates for each operation (Λ, Definition 5.3);
2. establish an assignment ($\lambda \in \Lambda$) for each operation using classical approaches [15];
3. perform a post-order visit of the query plan extending the plan with encryption and decryption operations (Definition 5.4);
4. establish keys for the query plan execution (Definition 6.1);
5. dispatch sub-queries to subjects, including, in the communication to each subject, the keys that the subject needs to perform encryption/decryption operations.

The sequence of steps above assumes encryption and decryption to have negligible impact on query costs/performance (e.g., if AES is used). When this is not the case (e.g., if an OPE scheme is used), Steps 2 and 3 have to be combined (as we did in our tool, see Section 7), extending traditional optimizers to take encryption and decryption operations into proper consideration for the analysis of query costs and/or performance.

As stated in Section 2, our authorization model does not distinguish among different kinds of encryption. The query optimizer can choose to apply different encryption schemes (e.g., deterministic or randomized encryption) depending on the operations to be executed on the encrypted values [10, 22]. We propose to adopt, for each attribute, the scheme providing highest protection, while supporting the operations to be executed on the attribute's encrypted values. For instance, if for an attribute no operation needs to be executed on encrypted values, randomized encryption is used, while if equality conditions need to be evaluated, deterministic encryption is used. Each attribute can be encrypted with a different encryption scheme and with a different key, the only constraint is that attributes that are involved together in some operations (i.e., attributes that belong to the same set in the equivalence set of the root's profile) need to be encrypted with the same key to enable the execution of the operations.

The query dispatch operates according to classical approaches, with the only difference that subjects may be communicated keys and they may need to execute, in addition to operations requested by query computation, also encryption and decryption operations. We assume each subject S involved in a query plan to have a private (pri_S), public (pub_S) key pair. The communication to each subject will

S	Receives ($\mathbf{req_S}$)	Performs ($\mathbf{q_S}$)

$\mathbb{Y}$	$[[q_{\mathbb{Y}},(\mathrm{P},k_{\mathrm{P}})]_{\mathrm{pri}_{\mathbb{U}}}]_{\mathrm{pub}_{\mathbb{Y}}}$	SELECT T,decrypt(P^k,k_{P}) AS P
		FROM $[\![\mathrm{req}_{\mathbb{X}}]\!]$
		WHERE P >100

$\mathbb{X}$	$[[q_{\mathbb{X}},\text{-}]_{\mathrm{pri}_{\mathbb{U}}}]_{\mathrm{pub}_{\mathbb{X}}}$	SELECT T,avg(P^k) AS P^k
		FROM $[\![\mathrm{req}_{\mathbb{H}}]\!]$ JOIN $[\![\mathrm{req}_{\mathbb{I}}]\!]$ ON S^k=C^k
		GROUP BY T

$\mathbb{H}$	$[[q_{\mathbb{H}},(\mathrm{S},k_{\mathrm{SC}})]_{\mathrm{pri}_{\mathbb{U}}}]_{\mathrm{pub}_{\mathbb{H}}}$	SELECT encrypt(S,k_{SC}),D,T
		FROM Hosp
		WHERE D='stroke'

$\mathbb{I}$	$[[q_{\mathbb{I}},(\mathrm{C},k_{\mathrm{SC}})(\mathrm{P},k_{\mathrm{P}})]_{\mathrm{pri}_{\mathbb{U}}}]_{\mathrm{pub}_{\mathbb{I}}}$	SELECT encrypt(C,k_{SC}),encrypt(P,k_{P})
		FROM Ins

Figure 8: Query dispatch for the plan in Figure 7(a)

be signed with the private key of the user and encrypted with the subject's public key. Having a sub-query signed allows the recipient to verify its authenticity and integrity. Encrypting a sub-query with the public key of the recipient supports confidentiality of the communication. Note however that the correctness of our approach does not depend on the simple protection of the communication. As a matter of fact, the definition of profiles does not make any assumption on the confidentiality of the query, which could potentially be known (of course with conditions operating on encrypted values when demanded by encryption operations in the plan). Figure 8 illustrates the query dispatch for the plan in Figure 7(a). The plan starts with the request from $\mathbb{U}$ to $\mathbb{Y}$ ($\mathrm{req}_{\mathbb{Y}}$), which will call the sub-query at $\mathbb{X}$ ($\mathrm{req}_{\mathbb{X}}$), which in turn will call the sub-queries at $\mathbb{H}$ ($\mathrm{req}_{\mathbb{H}}$) and $\mathbb{I}$ ($\mathrm{req}_{\mathbb{I}}$).

Note that our approach relies on the correct enforcement of authorizations throughout the query plan. Since the definition of the query plan is outside the control of the involved data authorities, the query optimizer has to be trusted for such an enforcement. Each data authority will perform a control at its side, before releasing the data to a third party, to check that the user is authorized for the released data. In fact, a user requesting query execution is required to be authorized to access all data that are input to the query, which correspond to the base relations. The user is then trusted to involve other authorized subjects.

We close this section with an observation on authorization enforcement. In the description of our approach, for simplicity, we have assumed the control of the authorizations holding for a given subject simply as a check against the set $\mathcal{P}_S$ ($\mathcal{E}_S$, resp.) summarizing the attributes that subject S is authorized to access plaintext (in encrypted form, resp.). While the realization of such a control directly against a global repository storing $\mathcal{P}_S$ and $\mathcal{E}_S$, for all subjects, can be possible, in real applications we can expect authorizations over the different relations to be stored in a distributed manner, like the relations are, and remain under the control of the respective data authorities. This distributed storage and management of authorizations is completely in line with our approach. As a matter of fact, a major advantage of the consideration of authorizations holding only on specific relations (no cross-relations/cross-authority authorizations) is that it simplifies authorizations specification and management and makes our solution completely independent from the approach adopted for storing and managing authorizations. For instance, a data authority can: *i)* publish its access control policy (which would then result publicly visible), or *ii)* respond to explicit authorization requests. The first approach can facilitate access to the policy, but entails its complete exposure. The second approach has instead the advantage of maintaining the whole policy confidential, providing only the responses to individual authorization checks. Our proposal is independent of the specific approach adopted and can work with both of them.

7. ECONOMIC BENEFITS

Given a query plan T(N) for a query q, there might exist a number of assignments (and of minimally extended plans) that satisfy all the authorizations. Among these plans, the user can choose the one optimizing a parameter of her interest such as cost or performance. In particular, we expect the economic cost to be the driving factor in the choice of the assignment of operations to candidates as long as performance remains above a given threshold. In our experiments, we then aimed to determine an assignment that minimizes the economic cost of query evaluation, which can possibly be combined with a threshold on the maximum performance overhead admitted for the evaluation of the query. We assume that the cost C_q of executing a query q is computed as $C_q = \sum_{n \in \texttt{N}} C^n_{\text{cpu}} + C^n_{\text{io}} + C^n_{\text{net\_io}}$, with N the nodes in the query plan, and $C^n_{\text{cpu}}, C^n_{\text{io}}, C^n_{\text{net\_io}}$ the cost, in USD, of cpu processing (time multiplied by cost per minute), local i/o (size of processed data multiplied by the unit cost of data access), and network i/o (size of transmitted data multiplied by the cost of network data transfer), respectively, for the execution of the operation at node n. This is in line with the price lists of cloud providers, which charge users based on their use of cpu time, local i/o, and network i/o. In the query processing domain, we expect the cpu processing and the network i/o to be the most significant cost components.

To evaluate the economic benefits of our approach in distributed query execution, we implemented it as a Java-based tool and performed a series of experiments using TPC-H (1 GB configuration), as it is the reference benchmark for testing solutions through complex queries. We note that although the 22 TPC-H queries do not use any udf, the consideration of udf in the queries can only further improve the economic benefits already visible over TPC-H. In fact, udfs are typically computationally-intensive, and therefore for queries including them the dominating cost factor is the cpu processing for evaluating the udfs. Delegating such computation to external advantageous providers enables a saving largely above the costs it adds for data transmission. The ability provided by our approach to delegate such computation to providers with the lowest cost among those trusted to access (in plaintext or encrypted form) the involved data can then bring considerable advantages since even small reductions in price lead to a reduction in the economic costs associated with the execution of these queries.

Our tool receives in input a database schema, a query plan, a description of the network configuration, and the authorizations. It provides as output the assignment of operators in the query plan to subjects, introducing the encryption and decryption steps needed to satisfy the authorizations. Input query plans are those produced by the PostgreSQL optimizer. The mapping from relational algebra operators (considered in our model) to the physical PostgreSQL operators was immediate, as every PostgreSQL operator has a natural correspondence to relational algebra operators.

Our implementation is based on a dynamic programming strategy to explore the possible assignments of candidates to operators in the query plan to identify the solution with minimum cost. We set the cost values input to the experiments for cloud providers based on the listings of the most common cloud providers on the market (e.g., Amazon S3, Google Compute Engine). We considered, as it is to be expected in the scenarios that motivate this research, a relatively high cost for the direct involvement of the user and of data authorities, which are 10 times and 3 times, respectively, the cpu processing cost of cloud providers. The estimate for the data authorities was based on price lists of government-backed organizations in Italy. The estimates of the size of the processed data and the processing time for the relational operators were those returned by the PostgreSQL optimizer. The tool assumed the adoption of four possible encryption techniques (randomized symmetric encryption; deterministic symmetric encryption; Pallier crypto-system; and an OPE scheme) and estimated the cost for their execution based on common benchmarks (AES has been used for the two variants of symmetric encryption, with essentially identical cost), represented in terms of computational effort. Our implementation also considered the increase in size that may derive from the application of encryption. The network configuration assumed the authorities controlling the data and the cloud providers to be connected by high-bandwidth (10Gbps) connections; the client was assumed to be connected to both with a lower-bandwidth (100Mbps) connection.

We considered the execution of the 22 TPC-H queries distributing the 8 TPC-H tables between two authorities and considering then the following three scenarios for the authorizations.

UA authorizations permit access to different base relations only to the user (issuing the query).

UAP$_{\text{enc}}$ cloud providers are authorized to access in encrypted form all the attributes of all the base relations.

UAP$_{\text{mix}}$ modifies the previous scenario with authorizations allowing cloud providers to access in plaintext half of the attributes that were previously only accessible in encrypted form.

Figure 9 illustrates the comparison of the different economic costs for the 22 TPC-H queries. Given the heterogeneity of the different queries and their cost, we report the cost in a normalized form considering, for each query, a unitary cost for **UA** (reported by the dotted line in the figure). As expected, compared with a base scenario (**UA**) where only the expensive user and data authorities can perform computations, the involvement of providers (**UAP$_{\text{enc}}$, UAP$_{\text{mix}}$**) enables significant savings. As a matter of fact, enabling involvement of other parties in the computation allows users to fully enjoy the economic benefit of the open cloud market. Among the scenarios including providers in the computation, **UAP$_{\text{mix}}$** clearly enjoys the most savings as plaintext visibility enables providers to execute all operations without the cost of encryption or the restrictions that derive from the access policy. Figure 10 illustrates the cumulative cost for the same queries illustrated in Figure 9. The results obtained show that involving providers in the processing of encrypted data (**UAP$_{\text{enc}}$** scenario) provides a saving of 54.2% compared to the base **UA** scenario. The

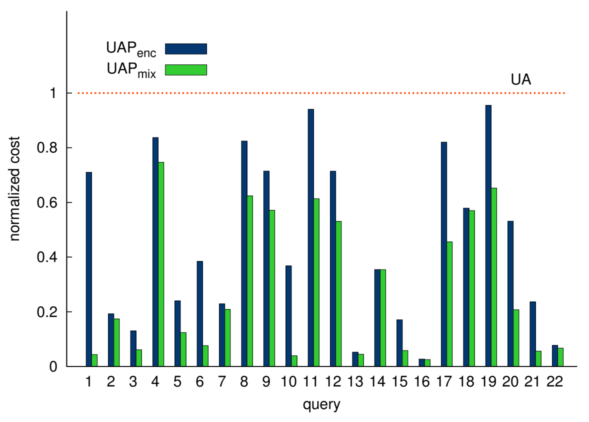


Figure 9: Economic cost of evaluating individual queries considering different authorization scenarios

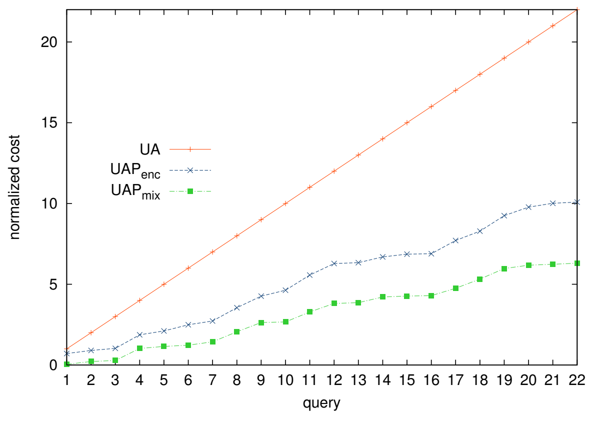


Figure 10: Total economic cost of evaluating queries considering different authorization scenarios

saving further increases (71.3%) with the loosening of the policy (**UAP$_{\text{mix}}$** scenario). We note that the saving is expected to be high when the difference in the prices of cloud providers is significant. Indeed, our approach permits to partially delegate operations running on encrypted data to cloud providers with economically convenient price lists, even if they are not trusted to access plaintext data. These benefits become even more evident in domains where only specific cloud providers can be adopted for the management of specific kinds of data (e.g., medical data) since they typically have higher price lists than the cloud providers operating in the open cloud market.

8. RELATED WORK

The problem of managing queries in distributed scenarios has been extensively studied, but traditional solutions (e.g., [15, 17]) as well as modern approaches that consider big data analytics (e.g., [2, 20]) do not take into consideration access restrictions. In the relational database context, access restrictions can be supported by views (e.g., [7, 13, 21]), access patterns (e.g., [4]), or data masking (e.g., [16]). Such proposals however do not consider encryption.

Work closest to ours has addressed the problem of protecting data confidentiality in distributed computations (e.g., [8, 18, 24]). In [24] the authors present an approach to collaboratively execute queries on data subject to access restrictions, considering different join evaluation strategies. In [8] the authors provide a solution for restricting access and sharing of distributed data, which supports the explicit consideration of join paths in the authorizations. The proposal in [18] aims to protect computations in hybrid clouds, preventing flows of sensitive information to the public cloud. These works confirm the relevance of the problem, but focus on different aspects. In particular, [24] considers only data explicitly exchanged among providers and do not take into consideration implicit information disclosure. While providing a more expressive authorization model, the approach in [8] requires collaborative specification of authorizations. None of the proposals considers the possibility of protecting data with encryption. Our proposal takes then a novel approach supporting different visibility levels over data and flexibly injecting encryption on-the-fly to protect data and enable the controlled involvement of cloud providers in the query computation. In [11] the authors address a complementary problem allowing users to specify confidentiality requirements in query evaluation to protect the objective of their queries to some providers.

Several works (e.g., [1, 14, 19, 22]) have investigated the use and support of encryption for the protection of data in storage or query execution. Other approaches (e.g., [3, 5]) proposed solutions for using secure multiparty computation in query evaluation, to keep both the input operands and the result secret to the party in charge of query evaluation. Specific works (e.g., [9]) have designed techniques to verify the integrity of query results computed by potentially untrusted providers. All these solutions are complementary to our proposal, which aims to leverage the availability of solutions supporting operation execution on encrypted data for enforcing authorizations and enabling the controlled involvement of providers in query execution.

9. CONCLUSIONS

We leverage the availability of emerging solutions supporting computation over encrypted data to provide a novel flexible approach enabling controlled query execution in the cloud. Our approach allows independent data authorities to make their data available for access and collaborative query execution, and enables users to execute queries over such data with selective and controlled involvement of external cloud providers. A main advantage of our approach is the flexibility in the assignment of query operations to providers, with on-the-fly insertion of encryption and decryption to adjust visibility of data as dictated by the authorizations. Our work leaves room for extensions, such as the consideration of scenarios where source relations are not stored at the corresponding data authority, but (possibly in encrypted form) at a third party.

10. ACKNOWLEDGMENTS

This work was supported in part by: the EC within the 7FP under grant agreement 312797 (ABC4EU) and within the H2020 under grant agreement 644579 (ESCUDO-CLOUD), and by NSF under grant IIP-1266147.

11. REFERENCES

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

[2] 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*, pages 1383–1394, Melbourne, Australia, May-June 2015.

[3] 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.

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

[5] 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, February 2009.

[6] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton. Mad skills: New analysis practices for big data. *PVLDB*, 2(2):1481–1492, 2009.

[7] 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.

[8] 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.

[9] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Efficient integrity checks for join queries in the cloud. *JCS*, 24(3):347–378, 2016.

[10] S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati. Practical techniques building on encryption for protecting and managing data in the cloud. In P. Ryan, D. Naccache, and J.-J. Quisquater, editors, *Festschrift for David Kahn*, pages 205–239. Springer, 2016.

[11] N. Farnan, A. Lee, P. Chrysanthis, and T. Yu. PAQO: Preference-aware query optimization for decentralized database systems. In *Proc. of ICDE*, pages 424–435, Chicago, IL, March–April 2014.

[12] P. Grofig et al. Experiences and observations on the industrial implementation of a system to search over outsourced encrypted data. In *Proc. of Sicherheit*, pages 115–125, Vienna, Austria, March 2014.

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

[14] H. Hacigümüs, B. Iyer, S. Mehrotra, and C. Li. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of SIGMOD*, pages 216–227, Madison, WI, June 2002.

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

[16] M. M. Kwakye and K. Barker. Privacy-preservation in the integration and querying of multidimensional data models. In *Proc of PST*, pages 255–263, Auckland, New Zealand, December 2016.

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

[18] K. Y. Oktay, M. Kantarcioglu, and S. Mehrotra. Secure and efficient query processing over hybrid clouds. In *Proc. of ICDE*, pages 733–744, San Diego, CA, April 2017.

[19] R. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *Proc. of SOSP*, pages 85–100, Cascais, Portugal, October 2011.

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

[21] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proc. of SIGMOD*, pages 551–562, Paris, France, June 2004.

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

[23] J. Vaidya. Privacy in the context of digital government. In *Proc. of DG.O*, pages 302–303, College Park, MD, June 2012.

[24] 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.