Distributed query evaluation over encrypted data

Sabrina De Capitani di Vimercati[1][0000−0003−0793−3551], Sara Foresti[1][0000−0002−1658−6734], Sushil Jajodia[2][0000−0003−3210−558X], Giovanni Livraga[1][0000−0003−2661−8573], Stefano Paraboschi[3][0000−0003−0399−1738], and Pierangela Samarati[1][0000−0001−7395−4620]

[1] Università degli Studi di Milano, Italy – *firstname.lastname*`@unimi.it`
[2] George Mason University, USA – `jajodia@gmu.edu`
[3] Università degli Studi di Bergamo, Italy – `parabosc@unibg.it`

Abstract. The availability of a multitude of data sources has naturally increased the need for subjects to collaborate for distributed computations, aimed at combining different data collections for their elaboration and analysis. Due to the quick pace at which collected data 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 a different encryption scheme and key) for their external storage. This makes distributed computations combining these sources hard. 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 it is considered economically convenient.

1 Introduction

Our society and economy more and more rely on the knowledge that can be generated by analysis and computations combining data that are produced and owned/controlled by different parties. The cloud, thanks to a variety of storage and computational providers with different costs and performance guarantees, represents an accelerator for such needs. Data owners can in fact outsource their data to storage providers, making them (selectively) available for computations with reduced management burden at their own side. At the same time, users requiring analysis can (partially) delegate expensive computations to computational providers, with clear performance and economic benefits [9]. However, there is no such thing as a free lunch, and the scenario can be complicated by the fact that 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.

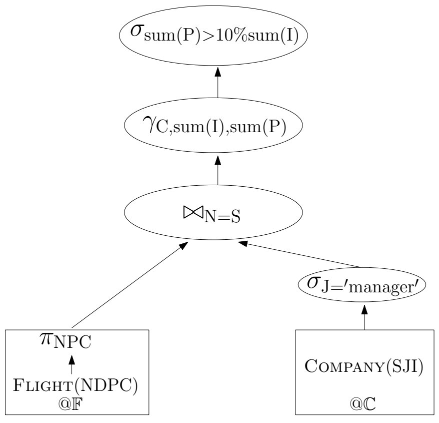
To solve this issue and ensure data protection while permitting the consideration of a large spectrum of providers for computations, a recent approach proposed a simple, yet flexible, authorization model that enriches the traditional yes/no visibility that a subject can have over data with a third visibility level, granting a subject visibility over encrypted versions of the data [9]. In this way, subjects that are economically convenient, but possibly not fully trusted for accessing data content, may still be involved in computations over encrypted data. To enforce the authorization policy, visibility over data is dynamically adjusted by inserting, before passing a dataset to a subject not trusted for plaintext access, on-the-fly encryption operations. Similarly, the encryption layer can be dynamically removed through on-the-fly decryption when requested for operations that cannot be executed over encrypted data.

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 providers that, while being economically convenient, cannot be considered fully trusted. Intuitively, the spectrum of potential providers that could be adopted for storing datasets could be enlarged if data are encrypted, by their owners, before outsourcing. 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. Users might in fact leverage economically convenient providers for the computation, and owners can outsource their datasets to economically convenient providers with the guarantee that their data will be improperly accessed neither in storage, nor in computation. The consideration of encrypted storage in collaborative computations brings however complications, since encryption in storage is not specifically inserted 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 (Section 2). Second, we identify the need, and propose a solution 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 (Section 3). Third, we provide an approach for computing an economically convenient assignment of computation operations to subjects in complete obedience of authorizations (Section 4). We discuss related works in Section 5 and conclude the paper in Section 6.

2 Relation profiles and authorizations

We consider a scenario characterized by three kinds of subjects: *1) data authorities*, each owning one or more relational tables possibly stored at external



$\sigma_{\mathrm{sum(P)>10\%sum(I)}}$
$\gamma_{\mathrm{C,sum(I),sum(P)}}$
$\bowtie_{\mathrm{N=S}}$
$\sigma_{\mathrm{J='manager'}}$
π_{NPC}
Flight(NDPC)
@$\mathbb{F}$
Company(SJI)
@$\mathbb{C}$

(a) Query plan

$[\mathrm{NCP},\_]\to\mathbb{U}$
$[\mathrm{SJI},\_]\to\mathbb{U}$

$[\mathrm{PC},\mathrm{N}]\to\mathbb{X}$
$[\_,\mathrm{SJI}]\to\mathbb{X}$

$[\mathrm{DP},\mathrm{CN}]\to\mathbb{Y}$
$[\mathrm{JI},\mathrm{S}]\to\mathbb{Y}$

$[\mathrm{NC},\mathrm{P}]\to\mathbb{Z}$
$[\mathrm{S},\mathrm{JI}]\to\mathbb{Z}$

$[\_,\mathrm{NDPC}]\to\mathbb{F}$
$[\_,\mathrm{JI}]\to\mathbb{F}$

$[\_,\mathrm{NC}]\to\mathbb{C}$
$[\_,\mathrm{SJI}]\to\mathbb{C}$

$[\mathrm{NC},_]\to$any
$[_,\mathrm{SJ}]\to$any

(b) Authorizations

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

storage providers; *2) users*, submitting queries over relations under the control of different authorities; and *3) computational providers*, which can be involved for query evaluation. Queries can be of the general form "select from where group by having" and can include joins among relations under control of different data authorities. Execution of queries is performed according to a query plan established by the query optimizer, 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 base relations, after the projection of the subset of attributes of interest for the query. For simplicity, but without loss of generality, we assume that attributes in the relations have different names.

Example 1. Consider two data authorities, a flight company and a commercial company with one relation each: 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,I,J) reports the Social security number, Income, and Job of the company employees. These relations are stored in encrypted form at providers $\mathbb{F}$ and $\mathbb{C}$, respectively. The system is characterized by 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)>10%sum(I)", retrieving the classes for which the overall price of tickets is above the 10% of the income of the managers who bought such tickets. Figure 1(a) illustrates a plan for the query.

Relation profile. Besides the attributes included in its schema, a relation resulting from a computation can convey information on other attributes. The information content explicitly and implicitly conveyed by a (base or derived, that is, resulting from the evaluation of a sub-query) relation is captured by a *profile* associated with the relation. We extend the definition of relation profile in [9] to model the possible encrypted representation of attributes in storage.

Definition 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, and uses a scheme and a key (decided by the owning data authority) that do not change over time and are not shared among different data authorities. 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 need to 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 or 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). Hence, even if one of the attributes in the equivalence set has been projected out from the relation schema, its values are still conveyed by the presence of 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 in encrypted form, respectively, in the relation schema. The profile of a *derived* relation resulting from the evaluation of an operation depends on both the operation and the profile of the operand(s). Figure 2 illustrates the profiles resulting from the evaluation of relational algebra operators, and of encryption and decryption operations, which are peculiar of our model. 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 in R^{vp} and, on a gray background, R^{ve} and R^{vE}), i (implicit attributes in R^{ip} and, on a gray background, R^{ie}), and $\simeq$ (sets of equivalent attributes in $R^{\simeq}$ that have

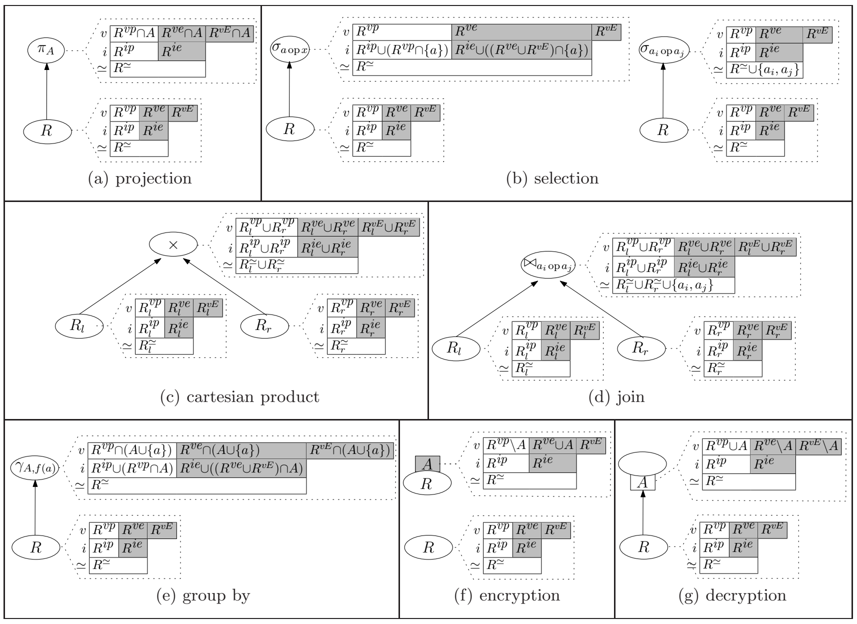


Fig. 2. Profiles resulting from relational, encryption, and decryption operations

been compared for R's computation). We represent encryption and decryption operations as gray and white boxes, respectively, containing the attributes to be encrypted/decrypted, attached to the operand relation or the resulting relation, respectively. Figure 3 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 are decrypted for enabling computations over them.

Authorizations. Authorizations aim at regulating 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 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 $R\setminus(P{\cup}E)$. Subject 'any' can be used to specify a default authorization applying to all subjects for which no authorization is 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

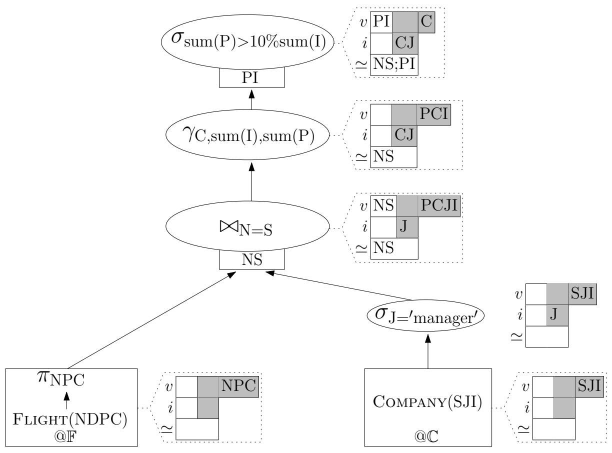


Fig. 3. Query plan with profiles

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 the 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 2. Figure 1(b) illustrates an example of a set of authorizations regulating access to relations Flight and Company of our running example. 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, computational providers $\mathbb{X}$, $\mathbb{Y}$, and $\mathbb{Z}$ have plaintext or encrypted visibility over a subset of the attributes in the two relations.

Authorization verification. To be authorized for a relation, a subject needs the 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 it 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 (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. Consider a relation R with profile [P,C,S,_,_,{IP}] and the authorizations in Figure 1(a). Provider $\mathbb{Z}$ is not authorized for the relation since it cannot access P in plaintext (Condition 1); $\mathbb{C}$ and $\mathbb{F}$ are not authorized 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 the relation.

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.

3 Extended minimum cost query plan

Given a query plan `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.

3.1 Candidates

Given a query plan `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 (a subset of) the attributes are encrypted in 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 of some attributes in the operand relation that she can access only in encrypted form.

Example 4. With reference to our running example, $\mathbb{Y}$ 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 for $\mathbb{Z}$ to be authorized for evaluating 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 who are authorized for evaluating the operation is possibly larger. In fact, encrypted attributes are also accessible by subjects with plaintext visibility. To determine candidates, 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. We note that the encryption of the attributes in the operands is always possible, since any attribute can be encrypted by the subject computing the operand (who can see it in plaintext). Similarly, 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.

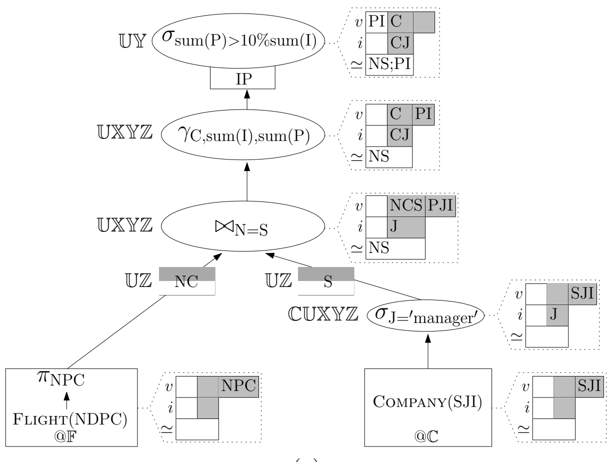
Definition 4 (Candidate). *Let* $\texttt{T(N)}$ *be a query plan,* $n \in \texttt{N}$ *be a non-leaf node, and* $n_l, n_r \in \texttt{N}$ *be its left and right child (if any),* $n.A_p$ *be the set of attributes that need to be in plaintext for the evaluation of* n*, and* $\mathcal{S}$ *be a set of subjects. 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);*
2) attributes in $n.A_p$ *in plaintext;*
3) n*, assuming the encryption of all the visible attributes in its operand(s) (Definition 3).*

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

Example 5. Figure 4(a) reports, for each node in the query plan of Example 1, the candidates who can evaluate the operation in 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 over their encrypted in storage representation; *ii)* the evaluation of the join and of the group by require the re-encryption of the involved attributes; and *iii)* the comparison of SUM(P) and SUM(I) can only be done over plaintext values.

The set of candidates along a query plan enjoys a nice *monotonicity* property. In fact, relation profiles never lose attributes, but can only gain new ones (see



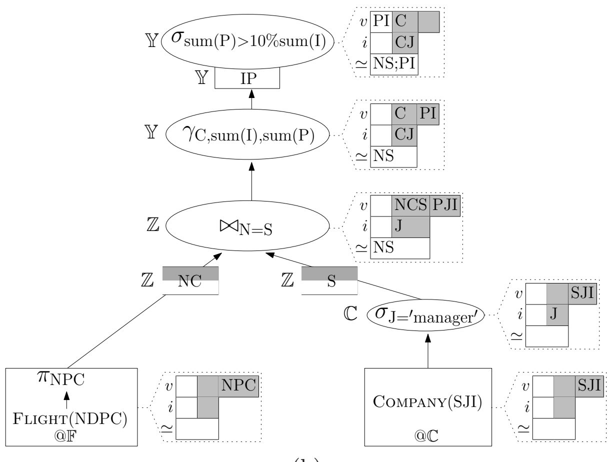


Fig. 4. Extended query plan with candidates (a) and with assignees (b)

Figure 3). Hence, a subject authorized for n is also authorized for its descendants in the query plan (the set of candidates monotonically decreases going up in the tree). This is true for all operations that do not require plaintext visibility over attributes, or which leave a trace in the implicit component of the resulting relation profile. A query plan `T(N)` is extended with encryption, decryption, and re-encryption, generating an *extended query plan*, denoted `T′(N′)`. Figure 4(a) illustrates an example of an extended version of the query plan in Figure 1(a), where attributes NC and S are re-encrypted (graphically represented with the gray and white rectangles below the join node), and attributes IP are decrypted.

Encryption, decryption, and re-encryption are used to adjust visibility and guarantee correct authorization enforcement. Their injection depends on the subjects to which operations are assigned. The injection of encryption and decryption operations does not affect the monotonicity property: the set of candidates of an encryption node corresponds to the one of the node to which encryption applies (i.e., of its child), and the set of candidates of a decryption node corresponds to the one of the node operating on the result of the decryption (i.e., of its parent). For example, candidates for the decryption of IP in Figure 4(a) are those for the selection to which decryption is connected (and hence are not explicitly reported in the figure). The consideration of re-encryption operations, necessary when the in-storage encryption scheme does not support operation execution, on the other hand, deserves a special treatment. 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. Indeed, 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 vive-versa). Note that the set of candidates for n_p is a subset of the candidates for n_c, since a candidate for n_p needs to have at least visibility on the relation produced by n_c. Figure 4(a) reports, for the two re-encryption operations, the set of candidates that could re-encrypt the involved attributes.

Given a query plan and the candidates for each of its nodes, it is then necessary to select, for each node, a subject (chosen among its candidates) in charge of the evaluation of the corresponding operation (i.e., the assignee of node n). Given a query plan, there can exist different possible assignments that respect authorizations and permit query execution. In the next section, we discuss how to determine an authorized assignment.

3.2 Authorized assignment and minimum cost query plan

Given a query plan $\mathtt{T(N)}$ and the set $\Lambda(n)$ of candidates for each node $n \in \mathtt{N}$, it is possible to determine an assignment of nodes to subjects taken from the corresponding set of candidates by inserting encryption and decryption operations. Such an assignment exists if, 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 same base relation in encrypted or plaintext form. Encryptions are inserted to enforce authorizations, and decryptions are inserted to adjust attributes visibility for operation evaluation, and are attached to the node requiring each of them. These operations can be performed by the same subject assigned to the nodes to which encryption/decryption are attached. Re-encryption, on the contrary, could be assigned to a different subject, and can be inserted at any point in the query plan, before the node that represents the operation for which re-encryption is needed. We also note that, differently from encryption and decryption operations, 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 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 6. Consider attribute C of our running example, which needs to be re-encrypted for the evaluation of GROUP BY clause. For an authorized assignment, we need a subject who can access attributes N and P in encrypted form and C in plaintext. 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 can be formalized by the following theorem.

Theorem 1 (Existence of an authorized assignment). *Let* $\mathtt{T(N)}$ *be a query plan,* $\forall n{\in}\mathtt{N}$, $n.A_e$ *be the set of attributes that need to be re-encrypted for the evaluation of* n*,* $\mathcal{S}$ *be a set of subjects and,* $\forall n{\in}\mathtt{N}$, $\Lambda(n)$ *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* $\mathtt{T'(N')}$*, with* $\lambda(n) \in \Lambda(n)$*, that does not violate any authorization.*

We can then conclude that, if there exists an authorized assignment for the query plan, any combination of subjects chosen from the candidate sets of the nodes in the query plan can be made authorized by injecting encryption, decryption, and re-encryption operations. For instance, Figure 4(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 plan; and *ii) data transfer cost* for the relations exchanged between subjects for query evaluation. The cost of query evaluation is obtained by summing these two cost components, taking into consideration also the 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 1 (Minimum cost query plan). Let $\mathtt{T(N)}$ be a query 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:

1. $\forall n {\in} \mathtt{N}'$, $\lambda(n) \in \Lambda(n)$, that is, the subject in charge of the evaluation of a node is one of its candidates;
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 $\mathit{cost}(\mathtt{T}'', \lambda') {<} \mathit{cost}(\mathtt{T}', \lambda)$

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

4 Computing assignment

The proposed heuristics operates in three phases (see Figure 5). The first phase identifies the set of candidates associated with the nodes of the query plan given as input. The second phase chooses, for each operation in the query plan, the subject (among the corresponding candidates) in charge of its execution, and inserts the needed re-encryption operations. The third phase inserts the encryption and decryption operations. The procedures corresponding to these phases are presented in Figures 5, 6, and 7 and illustrated 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 child and right child, respectively.

Identify candidates. Recursive procedure **Identify_Candidates** (Figure 5) performs a post-order visit of the query plan to identify, for each node, the candidates for its evaluation. For each node n, the procedure computes its profile, assuming that all the attributes in the operands are encrypted unless demanded for the evaluation of n (lines 8-12). The procedure then determines the candidates for n, checking among the candidates of n's operands or, for operations operating on plaintext attributes that do not leave a trace in the implicit component, also among the other subjects (lines 15-21). Note that the set of candidates for leaf nodes is set to the complete set of subjects (line 6), even if leaf nodes are assigned to the storing provider, to simplify the computation of the candidate sets in the query plan. For simplicity, but without loss of generality, we assume all the attributes in base relations to be encrypted in storage. 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 7, 13-14).

```
MAIN(T(N), $\mathcal{S}$)
1:  Compute_Cost(T.$root$)
2:  insert a node $client$ as parent of T.$root$ assigned to the user $\mathbb{U}$ formulating the query
3:  Identify_Candidates(T.$root$) /* Step 1: identify candidates */
4:  $to\_enc\_dec$=$\emptyset$
5:  Compute_Assignment(T.$root$) /* Step 2: compute assignment and inject re-encryption */
6:  Extend_Plan(T.$root$) /* Step 3: inject encryption/decryption */

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

Fig. 5. Pseudocode of our heuristic algorithm and of procedure **Identify_Candidates**

Choose assignment. Recursive procedure **Compute_Assignment** (Figure 6) performs a pre-order visit of the query 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), or move 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 being candidate of the node. 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 15-16) applies only when n is assigned to a subject S different from its parent and is computed as the product between the estimated size of the relation generated by n and the transfer cost of the subject in charge of evaluating n (in line with cloud market price lists, we consider only outbound traffic).
– *Computational cost* (line 18) is the sum of the costs of evaluating all the nodes in the subtree rooted at n by subject S. Such a cost is pre-computed by recursive procedure **Compute_Cost**, which visits the query plan in pre-order summing the cost of the evaluation of the subtrees rooted at the children of n with the cost of evaluating n, which is obtained by multiplying the estimated computation complexity of evaluating n in $n.TotA_e$ and $n.TotA_p$ by the computation price of S. The costs precomputed by procedure **Com-**

```
**Compute_Assignment**($n$)
1:  $S_{min}$=NULL
2:  $min=+\infty$
3:  if $n_l=n_r$=NULL /* $n$ is a leaf node */
4:  then  $\lambda(n)=n.S$ /* storage provider for the corresponding relation */
5:        if  $to\_enc\_dec \cap R \neq \emptyset$
6:        then insert a re-encrypt node $new$ for $to\_enc\_dec \cap n.vE$ as parent of $n$
7:             $\Lambda(new)=\{S \in \mathcal{S}$: $S$ is authorized for $n$ and to access $to\_enc\_dec \cap n.vE$ in plaintext$\}$
8:             for each $S \in \Lambda(new)$ do
9:                 $cost = (dec\_cost(to\_enc\_dec)+enc\_cost(to\_enc\_dec)) \cdot S.comp\_price+$
10:                       $+\ n.size \cdot (S.transf\_price+\lambda(n).transf\_price)$
11:                if $cost<min$
12:                then  $min=cost$, $S_{min}=S$
13:            $\lambda(new)=S_{min}$
14: else  if $n$ is not a re-encryption operation
15:       then for each $S \in \Lambda(n)$ do
16:               if $S \neq \lambda(n_p)$ then $cost=n.size \cdot S.transf\_price$ /* transfer cost */
17:               else $cost=0$ /* transfer cost */
18:               $cost = cost+comp\_cost[n,S]$ /* computational cost */
19:               for each $a \in (n.TotA_p \cup n.TotA_e) \cap \mathcal{P}_S$ do /* $S$ decrypts the attribute */
20:                   $cost=cost+dec\_cost(a) \cdot S.comp\_price$
21:               for each $a \in (n.TotA_e \setminus \mathcal{P}_S)$ do /* need to delegate re-encrypt of $a$ */
22:                   $cost = cost+(dec\_cost(a)+enc\_cost(a)) \cdot avg\_comp\_price+$
23:                           $a.size(avg\_transf\_price+S.transf\_price)$
24:               for each $a \in (to\_enc\_dec \cap \mathcal{P}_S)$ do /* $S$ can re-encrypt $a$ */
25:                   $cost=cost+(dec\_cost(a)+enc\_cost(a)) \cdot S.comp\_price$
26:               if $cost<min$
27:               then  $min=cost$
28:                     $S_{min}=S$
           /* select the subject in charge of the evaluation of $n$ */
29:        $\lambda(n)=S_{min}$
30:        if $to\_enc\_dec \cap \mathcal{P}_{\lambda(n)} \neq \emptyset$
31:        then  insert a re-encrypt node $new$ for $to\_enc\_dec \cap \mathcal{P}_{\lambda(n)}$ as parent of $n$
32:              $\lambda(new)=\lambda(n)$
33:              $to\_enc\_dec=to\_enc\_dec \setminus \mathcal{P}_{\lambda(n)}$
34:        $to\_enc\_dec=to\_enc\_dec \cup (n.A_e \setminus \mathcal{P}_{\lambda(n)})$ /* delegated re-encryption */
35:        if $n.A_e \cap \mathcal{P}_{\lambda(n)} \neq \emptyset$
36:        then  insert a re-encrypt node $new$ for $n.A_e \cap \mathcal{P}_{\lambda(n)}$ as child of $n$
37:              $\lambda(new)=\lambda(n)$
38: if $n_l \neq$NULL then **Compute_Assignment**($n_l$)
39: if $n_r \neq$NULL then **Compute_Assignment**($n_r$)

**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:      $comp\_cost[n,S] = comp\_cost[n_l,S] + comp\_cost[n_r,S] + n.comp\_cost \cdot S.comp\_price$
```

Fig. 6. Pseudocode of procedures **Compute_Assignment** and **Compute_Cost**

pute_Cost are stored in a matrix, *comp_cost*[n,S], with a row for each node and a column for each subject.

– *Decryption cost* (lines 19-20) 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 computation price of S.
– *Re-encryption cost* (lines 21-25) includes the cost of re-encryption operations performed by S as well as of re-encryption operations necessary to S for the evaluation of n but that need to be delegated to a different subject.

To keep track of the attributes that require re-encryption, we use variable *to_enc_dec*, which keeps track of 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 assumes that S will take care of their re-encryption (lines 24-25). If S needs to operate on an attribute encrypted on-the-fly on which she does not have plaintext visibility, 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 computation 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 21-23).

Among the candidates for the node, procedure **Compute_Assignment** selects the subject S_{min} with minimum estimated cost (line 29). 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 she is authorized to access in plaintext (lines 30-33), and these attributes are removed from *to_enc_dec*. 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 34). 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 35-37). 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 assigned to a subject upon injection in the tree, procedure **Compute_Assignment** does not need to operate over them.

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 (line 3-4). We note however that, when the visit reaches a leaf node, it is necessary to verify whether *to_enc_dec* is empty. If *to_enc_dec* is not empty, it is necessary to insert a re-encryption operation for the attributes in *to_enc_dec*, which is assigned to the less expensive subject who can access attributes in *to_enc_dec* in plaintext (lines 5-13). The need to involve a subject only for re-encryption operations happens only if no subject assigned to other operations in the query plan can access the attribute(s) of interest in plaintext.

Extend query plan. Recursive procedure **Extend_Plan** (Figure 7) 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 operation (as child of n and assigned to $\lambda(n)$) for those attributes that must be in plaintext for the evaluation of n but that are encrypted in its operands. The procedure also injects an encryption operation (as parent of n and assigned to $\lambda(n)$) for the attributes appearing in plaintext in the profile of n and that the assignee of n_p can access only in encrypted form (lines 6-

```
Extend_Plan($n$)
1:  if $n_l \neq$ NULL then Extend_Plan($n_l$)
2:  if $n_r \neq$ NULL then Extend_Plan($n_r$)
3:  if $n$=T.$root$
4:  then  insert a decryption node $new$ for $n.ve \cup n.vE$ as parent of $n$
5:         $\lambda(new)=\mathbb{U}$
6:  else  if $n_l \neq$ NULL AND $n.A_p \setminus n_l.vp \neq \emptyset$
7:        then insert a decryption node $new$ for $n.A_p \setminus n_l.vp$ as parent of $n_l$
8:        if $n_r \neq$ NULL AND $n.A_p \setminus n_r.vp \neq \emptyset$
9:        then insert a decryption node $new$ for $n.A_p \setminus n_r.vp$ as parent of $n_r$
10:       if $\mathcal{E}_{\lambda(n_p)} \cap n.vp \neq \emptyset$ then insert an encryption node $new$ for $\mathcal{E}_{\lambda(n_p)} \cap n.vp$ as parent of $n$
11:       $\lambda(new)=\lambda(n)$
12: Compute_Profile($new$); Compute_Profile($n$); Compute_Profile($n_p$)
```

Fig. 7. Pseudocode of procedure **Extend_Plan**

11). The procedure finally updates the profiles of the nodes impacted by the encryption/decryption operation (line 12).

Example 7. Considering the query plan and authorizations in Figure 1, the algorithm first visits the tree in post-order and identifies the candidates for each node (Figure 4(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 4(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 C$\in n.A_e$ in plaintext, C is inserted into *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 in 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 do so, $\mathbb{Z}$ decrypts and re-encrypts also S and N. We note that $\mathbb{Z}$ can evaluate the 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.

The algorithm illustrated in this section represents a heuristic approach for solving Problem 1 and operates in $O(|\mathtt{N}| \cdot |\mathcal{S} \cdot |\mathcal{A}|)$ time, with $\mathcal{A}$ the set of attributes involved in the query.

5 Related work

Traditional solutions aimed at distributed query evaluation and data analytics do not take into consideration access restrictions (e.g., [2, 4, 15, 17, 20]). Solutions aimed at enforcing access restrictions in the relational database scenario (e.g., view based access control [8, 13, 21], access patterns [3, 6], data masking [16]) instead do not consider encryption as a solution for protecting confidentiality.

The use of encryption for protecting data confidentiality, while supporting query evaluation, has been widely studied (e.g., [1, 14, 19, 24]). Alternative solutions studied the adoption of secure multiparty computation (e.g., [5, 7]) and of trusted hardware components (e.g., [23]) to support query evaluation. All these solutions are complementary to our work, which can rely on these techniques to partially delegate query evaluation over encrypted data to subjects who are not authorized for plaintext visibility over (a subset of) the attributes.

Recent works have addressed the problem of protecting data confidentiality in distributed computation. The proposed solutions aim at controlling (explicit and implicit) information flows among subjects as a consequence of distributed computations (e.g., [10, 18, 22, 25]). The work closest to ours is represented by the solution in [9], on which our proposal builds. Indeed, the approach proposed in [9] for distributed query evaluation under access restrictions first proposed the idea of distinguishing between plaintext and encrypted visibility over the data, to the aim of enabling the delegation of computations over encrypted data to non-fully trusted subjects. This authorization model has been integrated into a real world query optimizer in [11]. The work in [9] 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, based on the needs for query evaluation. Our proposal extends such an approach to consider the more general scenario where base relations might be stored at an external provider, possibly in encrypted form.

In [12] 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.

6 Conclusions

We proposed an approach for leveraging storage and computational providers to enable distributed query execution, combining data possibly stored in encrypted form at external storage providers. Our solution allows data authorities to delegate the storage of their data to external providers, while still enabling collaborative query evaluation, selectively involving computational providers to limit the costs of query evaluation. 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.

Acknowledgements

This work was supported in part by the EC within the H2020 Program under projects MOSAICrOWN and MARSAL, by the Italian Ministry of Research within the PRIN program under project HOPE, and by JPMorgan Chase & Co under project “k-anonymity for AR/VR and IoT/5G”.

References

1. Agrawal, R., Asonov, D., Kantarcioglu, M., Li, Y.: Sovereign joins. In: Proc. of ICDE. Atlanta, GA (April 2006)
2. Alkowaileet, W., Alsubaiee, S., Carey, M., Li, C., Ramampiaro, H., Sinthong, P., Wang, X.: End-to-end machine learning with apache asterixdb. In: Proc. of DEEM. Houston, TX, USA (June 2018)
3. Amarilli, A., Benedikt, M.: When can we answer queries using result-bounded data interfaces? In: Proc. of PODS. Houston, TX, USA (June 2018)
4. Armbrust, M., Xin, R.S., Lian, C., Huai, Y., Liu, D., Bradley, J.K., Meng, X., Kaftan, T., Franklin, M.J., Ghodsi, A., Zaharia, M.: Spark SQL: Relational data processing in Spark. In: Proc. of SIGMOD. Melbourne, Australia (May-June 2015)
5. Bater, J., Elliott, G., Eggen, C., Goel, S., Kho, A., Duggan, J.: SMCQL: Secure query processing for private data networks. PVLDB 10(6), 673–684 (2017)
6. Benedikt, M., Leblay, J., Tsamoura, E.: Querying with access patterns and integrity constraints. PVLDB 8(6), 690–701 (2015)
7. Chow, S.S., Lee, J.H., Subramanian, L.: Two-party computation model for privacy-preserving queries over distributed databases. In: Proc. of NDSS. San Diego, CA, USA (February 2009)
8. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., Samarati, P.: Fragmentation in presence of data dependencies. IEEE TDSC 11(6), 510–523 (2014)
9. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., Samarati, P.: An authorization model for multi-provider queries. PVLDB 11(3), 256–268 (2017)
10. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Authorization enforcement in distributed query evaluation. JCS 19(4), 751–794 (2011)
11. Dimitrova, E., Chrysanthis, P., Lee, A.: Authorization-aware optimization for multi-provider queries. In: Proc. of SAC. Limassol, Cyprus (April 2019)
12. Farnan, N., Lee, A., Chrysanthis, P., Yu, T.: PAQO: Preference-aware query optimization for decentralized database systems. In: Proc. of ICDE. Chicago, IL, USA (March–April 2014)
13. Guarnieri, M., Basin, D.: Optimal security-aware query processing. PVLDB 7(12), 1307–1318 (2014)
14. Hacigümüs, H., Iyer, B., Mehrotra, S., Li, C.: Executing SQL over encrypted data in the database-service-provider model. In: Proc. of SIGMOD. Madison, WI, USA (June 2002)
15. Kossmann, D.: The state of the art in distributed query processing. ACM CSUR 32(4), 422–469 (2000)
16. Kwakye, M.M., Barker, K.: Privacy-preservation in the integration and querying of multidimensional data models. In: Proc of PST. Auckland, New Zealand (December 2016)
17. Levy, A.Y., Srivastava, D., Kirk, T.: Data model and query evaluation in global information systems. JIIS 5(2), 121–143 (1995)
18. Oktay, K.Y., Kantarcioglu, M., Mehrotra, S.: Secure and efficient query processing over hybrid clouds. In: Proc. of ICDE. San Diego, CA, USA (April 2017)
19. Popa, R., Redfield, C., Zeldovich, N., Balakrishnan, H.: CryptDB: Protecting confidentiality with encrypted query processing. In: Proc. of SOSP. Cascais, Portugal (October 2011)

20. Rheinländer, A., Leser, U., Graefe, G.: Optimization of complex dataflows with user-defined functions. ACM CSUR 50(3), 38:1–38:39 (2017)
21. Rizvi, S., Mendelzon, A., Sudarshan, S., Roy, P.: Extending query rewriting techniques for fine-grained access control. In: Proc. of SIGMOD. Paris, France (June 2004)
22. Salvaneschi, G., Köhler, M., Sokolowski, D., Haller, P., Erdweg, S., Mezini, M.: Language-integrated privacy-aware distributed queries. Proc. ACM Program. Lang. 3 (2019)
23. Sharma, S., Burtsev, A., Mehrotra, S.: Advances in cryptography and secure hardware for data outsourcing. In: IEEE ICDE. Dallas, TX, USA (April 2020)
24. Tu, S., Kaashoek, M., Madden, S., Zeldovich, N.: Processing analytical queries over encrypted data. PVLDB 6(5), 289–300 (2013)
25. Zeng, Q., Zhao, M., Liu, P., Yadav, P., Calo, S., Lobo, J.: Enforcement of autonomous authorizations in collaborative distributed query evaluation. IEEE TKDE 27(4), 979–992 (2015)