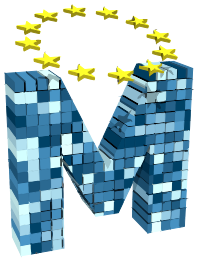
Ref. Ares(2021)6716189 - 31/10/2021



**Project title:** Multi-Owner data Sharing for Analytics and Integration respecting Confidentiality and OWNer control
**Project acronym:** MOSAICrOWN
**Funding scheme:** H2020-ICT-2018-2
**Topic:** ICT-13-2018-2019
**Project duration:** January 2019 – December 2021

# D4.4

# Report on Encryption-based Policy Enforcement and Controlled Query Execution

Editors: Sabrina De Capitani di Vimercati (UNIMI)
Flora Giusto (MC)
Reviewers: Alan Barnett (EISI)
Aidan O Mahony (EISI)
Jonas Böhler (SAP SE)

**Abstract**

This deliverable describes the techniques developed in WP4 for protecting data and computations in the digital data market. In particular, the document focuses on the techniques developed for supporting controlled data sharing for collaborative queries and fine-grained data retrieval. The model and techniques described enable the involvement of the data market in the execution of queries over data stored either in plaintext or encrypted form, possibly delegating parts of the execution to external parties when economically convenient while fully respecting the security policies associated with data. During the query execution, data can be dynamically wrapped and un-wrapped to regulate visibility over them. Fine-grained data retrieval is instead supported through the definition of a multi-dimensional index that is robust against inference exposure and supports the efficient execution of point and range queries over encrypted data. The experimental evaluation over a dataset with 3.2M data items shows that the multi-dimensional index performs well for query execution and requires limited storage at the client-side.

| Type | Identifier | Dissemination | Date |
|---|---|---|---|
| Deliverable | D4.4 | Public | 2021.10.31 |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825333.

# MOSAICrOWN Consortium

| | | | |
|---|---|---|---|
| 1. | Università degli Studi di Milano | UNIMI | Italy |
| 2. | EMC Information Systems International | EISI | Ireland |
| 3. | Mastercard Europe | MC | Belgium |
| 4. | SAP SE | SAP SE | Germany |
| 5. | Università degli Studi di Bergamo | UNIBG | Italy |
| 6. | GEIE ERCIM (Host of the W3C) | W3C | France |

**Disclaimer**: The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The below referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law. Copyright 2021 by Università degli Studi di Bergamo, Università degli Studi di Milano.

# Versions

| Version | Date | Description |
|---|---|---|
| 0.1 | 2021.10.05 | Initial Release |
| 0.2 | 2021.10.26 | Second Release |
| 1.0 | 2021.10.31 | Final Release |

# List of Contributors

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

| Chapter | Author(s) |
|---|---|
| Executive Summary | Sabrina De Capitani di Vimercati (UNIMI) |
| Chapter 1: Static and dynamic wrapping for collaborative computations | Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Giovanni Livraga (UNIMI), Stefano Paraboschi (UNIBG), Pierangela Samarati (UNIMI) |
| Chapter 2: Effective and efficient computations over wrapped data | Sabrina De Capitani di Vimercati (UNIMI), Dario Facchinetti (UNIBG), Sara Foresti (UNIMI), Gianluca Oldani (UNIBG), Stefano Paraboschi (UNIBG), Matthew Rossi (UNIBG), Pierangela Samarati (UNIMI) |
| Chapter 3: Conclusions | Sabrina De Capitani di Vimercati (UNIMI) |

# Contents

# List of Figures

# Executive Summary

This deliverable illustrates the advanced encryption-based techniques developed in MOSAICrOWN for supporting effective and efficient selective release, sharing, and use of data in collaborative scenarios. The problem has been investigated under different perspectives with the goal of guaranteeing protection while ensuring functionality. In particular, our solutions enable collaborative computations and fine-grained retrieval, within the data market, operating over wrapped (i.e., encrypted) data. The main content of this deliverable is organized in two chapters.

Chapter 1 presents an approach enabling collaborative computations over data encrypted in storage, selectively involving also external parties offering computational capabilities at competitive costs, while possibly not being fully trusted for complete access to plaintext data. This approach then permits enriching the functionality of the digital data market. A preliminary version of the techniques discussed in this chapter has been illustrated in Deliverables D4.1 and D4.2 [FL20, FP20]. Here, the approach has been extended to consider the more general scenario where base relations might be stored at an external provider, possibly in encrypted form. The consideration of encrypted storage in collaborative computations brings complications, since encryption in storage is not specifically applied according to the computations to be performed and may not support them, which could hence require additional decryption and re-encryption operations. The novel approach described in Chapter 1 then extends the definition of relation profile in [FP20] to model the possible encrypted representation of attributes in storage. The approach for computing the candidates authorized to execute a query operation has been redesigned. In addition to the consideration of encryption and decryption operations (i.e., the dynamic wrapping and unwrapping of data), the new approach also considers re-encryption operations that have to be possibly applied to guarantee the satisfaction of authorizations and to enable the evaluation of operations. Finally, a heuristic algorithm has been defined to the aim of effectively and efficiently finding an optimal assignment of query operations to subjects, fully respecting the security policies associated with data and such that the computed assignment has minimum economic cost.

Chapter 2 addresses the problem of supporting fine-grained access to wrapped data stored in the digital data market. In particular, the chapter presents an approach for indexing encrypted data stored at external providers to enable data market-side evaluation of queries. This approach supports the evaluation of point and range conditions on multiple attributes. Protection against inferences from indexes is guaranteed by clustering tuples in boxes that are then mapped to the same index values, so to ensure collisions for individual attributes as well as their combinations. The approach for index construction employs a spatial-based representation of tuples to be outsourced and a clever algorithm performing recursive cuts on such space resulting in a partitioning of tuples for indexes. Query translation and processing require the client to store a compact map. As confirmed by experimental evaluation, the indexing proposal provides for effective and efficient query evaluation, enjoying limited overhead and limited storage requirements at the client-side.

# 1. Static and dynamic wrapping for collaborative computations

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 data market, combined with the possibility of using 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 the data market, 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 the data market, with clear performance and economic benefits [DFJ+17]. However, 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 for the data market to rely on external 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, MOSAICrOWN has 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 [FL20, FP20]. 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 proposed authorization model 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 with 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. In this chapter, we then 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. The consideration of encrypted storage in collaborative computations brings, however, complications since encryption in storage is not specifically applied according to the computations to be performed and may not support them, which could hence require additional decryption and re-encryption operations.

The reminder of this chapter is organized as follows. Section 1.1 presents related works and the innovation of MOSAICrOWN. Section 1.2 re-defines the information flows enacted by a computation, necessary for authorization enforcement, based on the possibility of some data being stored

in encrypted form. Section 1.3 identifies the need, and proposes 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 1.4 illustrates an approach for computing an economically convenient assignment of computation operations to subjects in complete obedience of authorizations. Section 1.5 concludes the chapter.

## 1.1 State of the art and MOSAICrOWN innovation

We describe the state of the art and the innovation produced by MOSAICrOWN for collaborative computations over data stored both in plaintext and encrypted form in the data market and under the control of different authorities.

### 1.1.1 State of the art

Traditional solutions aimed at distributed query evaluation and data analytics do not take into consideration access restrictions (e.g., [AAC+18, AXL+15, Kos00, LSK95, RLG17]). Solutions aimed at enforcing access restrictions in the relational database scenario (e.g., view based access control [DFJ+14, GB14, RMSR04], access patterns [AB18, BLT15], data masking [KB16]) 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., [AAKL06, HIML02, PRZB11a, TKMZ13]). Alternative solutions studied the adoption of secure multiparty computation (e.g., [BEE+17, CLS09]) and of trusted hardware components (e.g., [SBM20]) 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., [DFJ+11, OKM17, SKS+19, ZZL+15]). The work closest to ours is represented by the solution in [DFJ+17], on which our proposal builds. Indeed, the approach proposed in [DFJ+17] 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 [DCL19]. The work in [DFJ+17] 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 [FLCY14] 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.

### 1.1.2 MOSAICrOWN innovation

MOSAICrOWN produced several advancements over the state of the art, which can be summarized as follows.

- The consideration of datasets involved in distributed computations that can also be stored in encrypted form (static wrapping) combined with an authorization model supporting dynamic wrapping of datasets. This benefits 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 store their datasets to economically convenient providers with the guarantee that their data will be improperly accessed neither in storage, nor in computation.
- The definition of a heuristics for computing an economically convenient assignment of computation operations to subjects in complete obedience of authorizations.

The approach for supporting static and dynamic wrapping for collaborative computations described in this chapter has been published in [DFJ+21]. A preliminary version of the authorization model has been describe in deliverables D4.1 [FL20] and D4.2 [FP20].

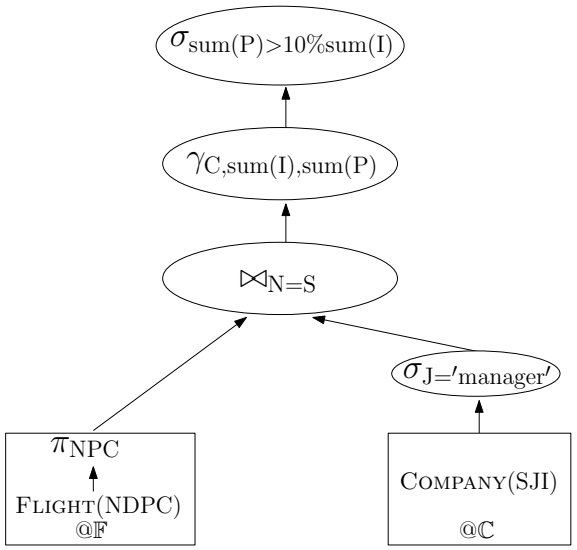
## 1.2 Relation profiles and authorizations

We consider a scenario characterized by three kinds of subjects: *1) data authorities*, each owning one or more relational tables possibly stored at external *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 1.2.1.** 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 1.1(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 [DFJ+17] to model the possible encrypted representation of attributes in storage.

**Definition 1.2.1 (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




(a) Query plan

[NCP,_]→𝕌
[SJI,_]→𝕌

[PC,N]→𝕏
[_,SJI]→𝕏

[DP,CN]→𝕐
[JI,S]→𝕐

[NC,P]→ℤ
[S,JI]→ℤ

[_,NDPC]→𝔽
[_,JI]→𝔽

[_,NC]→ℂ
[_,SJI]→ℂ

[NC,_]→any
[_,SJ]→any

(b) Authorizations

Figure 1.1: An example of a query plan (a) and of authorizations on relations FLIGHT and COMPANY stored at providers $\mathbb{F}$ and $\mathbb{C}$, respectively (b)

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 1.2 illustrates the profiles resulting from

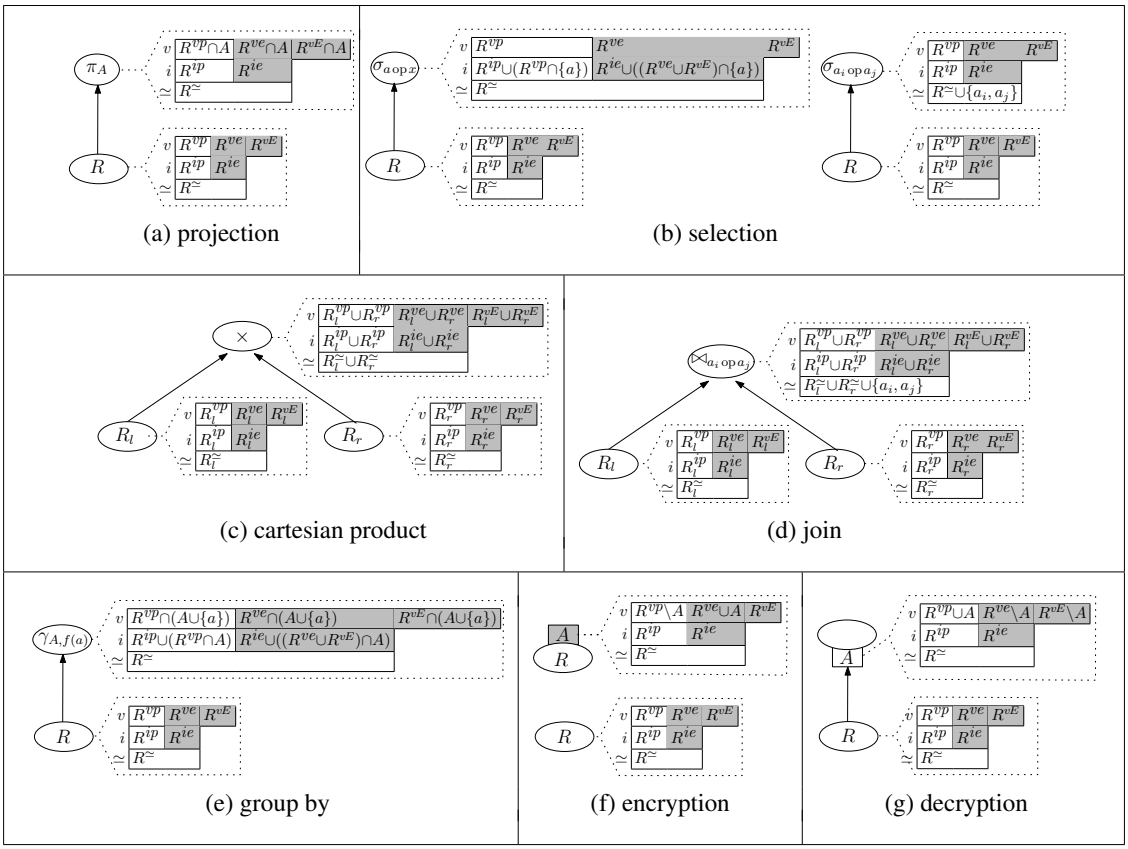



Figure 1.2: Profiles resulting from relational, encryption, and decryption operations

the evaluation of relational algebra operators, and of encryption and decryption operations, which are specific to 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 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 1.3 illustrates the profiles of the relations resulting from the evaluation of the operations in the query plan in Figure 1.1(a), assuming attributes NS and PI are decrypted for enabling computations over them. Note that, for simplicity, in the figure and in the remainder of this document, we denote a set of attributes simply with the sequence of the attributes composing it, omitting the curly brackets and commas.

**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 1.2.2 (Authorization).** Let $R$ be a relation and $\mathscr{S}$ 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 $S\in\mathscr{S}\cup\{\text{any}\}$.

Authorization $[P,E]\rightarrow S$ states that subject $S$ can access in plaintext attributes in $P$, in encrypted form attributes in $E$, and has no visibility over the attributes in $R\backslash(P\cup E)$. Subject 'any' can

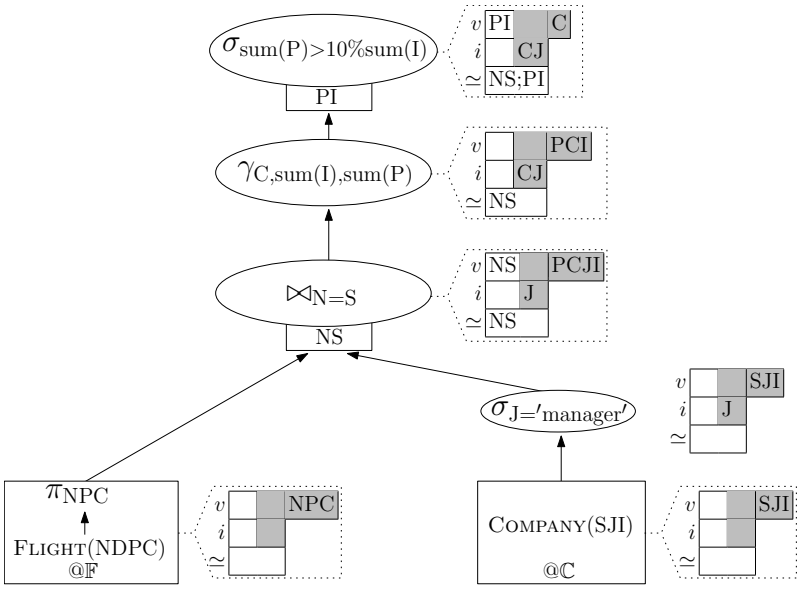



Figure 1.3: Query plan with profiles

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 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 1.2.2.** Figure 1.1(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 $\mathscr{P}_S$ ($\mathscr{E}_S$, 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 [DFJ+17] to take the two kinds of encryption into consideration.

**Definition 1.2.3 (Authorized Relation).** Let $R$ be a relation with profile $[R^{vp},R^{ve},R^{vE},R^{ip},R^{ie},R^{\simeq}]$. A subject $S \in \mathscr{S}$ is *authorized* for $R$ iff:

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

**Example 1.2.3.** Consider a relation $R$ with profile [P,C,S,_,_,{IP}] and the authorizations in Figure 1.1(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.

## 1.3 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.

### 1.3.1 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 1.3.1.** 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 1.3.1 (Candidate).** Let $\mathtt{T(N)}$ be a query plan, $n \in \mathtt{N}$ be a non-leaf node, $n_l, n_r \in \mathtt{N}$ be the left and right child (if any) of node $n$, $n.A_p$ be the set of attributes that need to be in plaintext for the evaluation of $n$, and $\mathscr{S}$ be a set of subjects. A subject $S \in \mathscr{S}$ 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 1.2.3);

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

3) $n$, assuming the encryption of all the visible attributes in its operand(s) (Definition 1.2.3).

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

**Example 1.3.2.** Figure 1.4(a) reports, for each node in the query plan of Example 1.2.1, 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 Figure 1.3). 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 $\mathtt{T(N)}$ is extended with encryption, decryption, and re-encryption, generating an *extended query plan*, denoted $\mathtt{T'(N')}$. Figure 1.4(a) illustrates an example of an extended version of the query plan in Figure 1.1(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 1.4(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

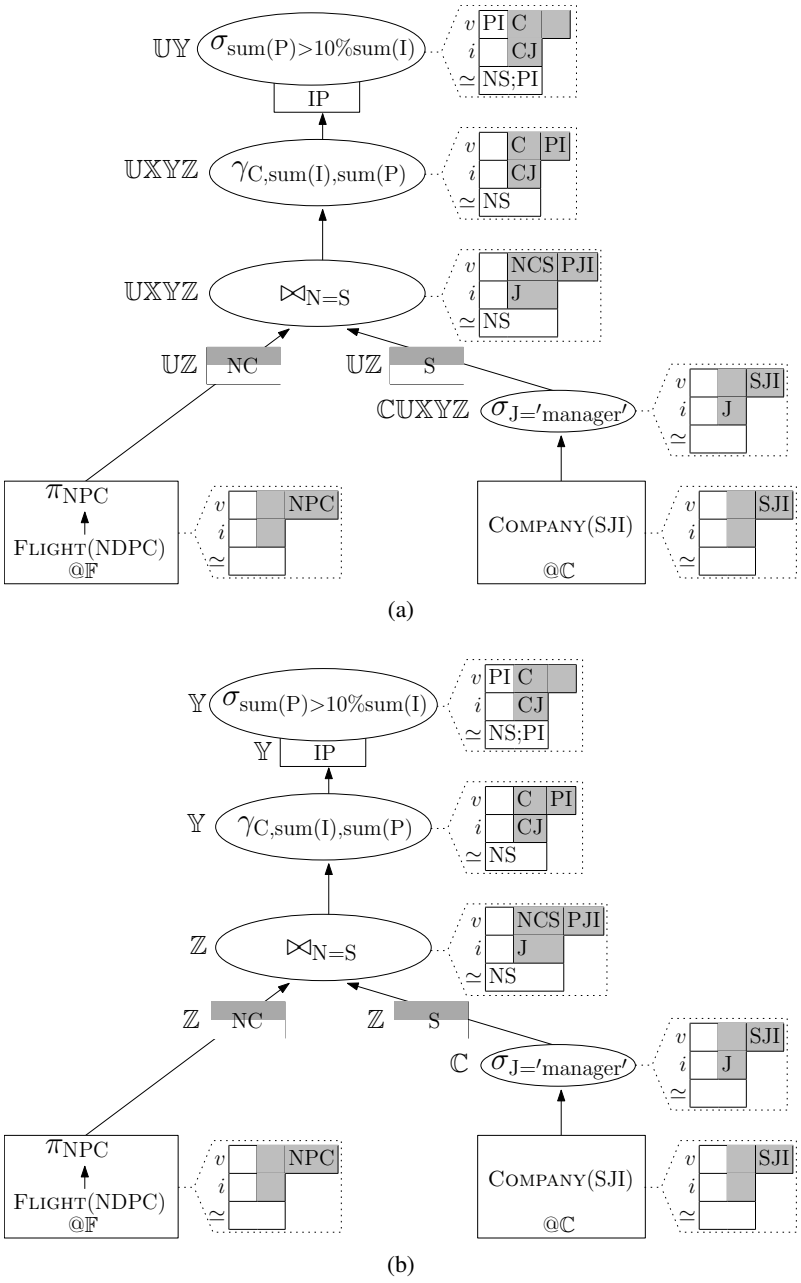



Figure 1.4: Extended query plan with candidates (a) and with assignees (b)

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 vice-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 1.4(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.

### 1.3.2 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. Encryption operations are inserted to enforce authorizations, and decryption operations 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 1.3.3.** 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 1.3.1 (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 $n$, $\mathscr{S}$ 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 $S \in \mathscr{S}$ s.t. $a \in \mathscr{P}_S$ and $R \subseteq \mathscr{P}_S \cup \mathscr{E}_S$, 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 $\lambda : \mathtt{N'} \rightarrow \mathscr{S}$ 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 1.4(b) illustrates an extended query plan that makes the assignment on the left of each node authorized according to the authorizations in Figure 1.1(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 1.3.1 (Minimum cost query plan).** Let $\mathtt{T}(\mathtt{N})$ be a query plan and $\mathscr{S}$ be a set of subjects. Determine an extended query plan $\mathtt{T}'(\mathtt{N}')$ of $\mathtt{T}$ and an assignment $\lambda : \mathtt{N}' \rightarrow \mathscr{S}$ 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. $\nexists \mathtt{T}''$, $\lambda'$ such that $\mathtt{T}''$ is an extended query plan of $\mathtt{T}$ and $\lambda'$ an assignment for $\mathtt{T}''$ such that $\forall n \in \mathtt{N}'$, $\lambda'(n) \in \Lambda(n)$ and $cost(\mathtt{T}'', \lambda') < cost(\mathtt{T}', \lambda)$

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

## 1.4 Computing assignment

The proposed heuristics operates in three phases (see Figure 1.5). 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 1.5, 1.6, and 1.7 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 1.5) 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

**MAIN**(T (N) , $\mathscr{S}$)

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: $\Lambda(n) = \mathscr{S}$ /* 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**($n_l \setminus n.A_p$, **decrypt**($n.A_p \cup n.A_e$, $n_l$))
11: $n_r =$**encrypt**($n_r \setminus n.A_p$, **decrypt**($n.A_p \cup n.A_e$, $n_r$))
12: **Compute_Profile**($n$) /* compute the relation profile according to Figure 1.2 */
13: $n.TotA_p = n.A_p \cup n_l.TotA_p \cup n_r.TotA_p$
14: $n.TotA_e = n.A_e \cup n_l.TotA_e \cup n_r.TotA_e$
15: $\Lambda(n) = \emptyset$
16: **if** $n_l.A_p \cup n_r.A_p \subseteq n.ip$
17: **then** $Cand = \Lambda(n_l) \cup \Lambda(n_r)$
18: **else** $Cand = \mathscr{S}$
19: **for each** $S \in Cand$ **do**
20: **if** $S$ is authorized for $n_l$, $n_r$, $n$
21: **then** $\Lambda(n) = \Lambda(n) \cup \{S\}$

Figure 1.5: Pseudocode of our heuristic algorithm and of procedure **Identify_Candidates**

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).

**Choose assignment**. Recursive procedure **Compute_Assignment** (Figure 1.6) 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).

**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: $\Lambda(new)$=$\{S \in \mathscr{S}$: $S$ 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** $a \in (n.TotA_p \cup n.TotA_e) \cap \mathscr{P}_S$ **do** /* $S$ decrypts the attribute */
20: $cost=cost+dec\_cost(a) \cdot S.comp\_price$
21: **for each** $a \in (n.TotA_e \setminus \mathscr{P}_S)$ **do** /* need to delegate re-encrypt of $a$ */
22: $cost = cost + (dec\_cost(a)+enc\_cost(a)) \cdot avg\_comp\_price +$
23: $a.size \cdot (avg\_transf\_price + S.transf\_price)$
24: **for each** $a \in (to\_enc\_dec \cap \mathscr{P}_S)$ **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** $to\_enc\_dec \cap \mathscr{P}_{\lambda(n)} \neq \emptyset$
31: **then** insert a re-encrypt node $new$ for $to\_enc\_dec \cap \mathscr{P}_{\lambda(n)}$ as parent of $n$
32: $\lambda(new)=\lambda(n)$
33: $to\_enc\_dec=to\_enc\_dec \setminus \mathscr{P}_{\lambda(n)}$
34: $to\_enc\_dec=to\_enc\_dec \cup (n.A_e \setminus \mathscr{P}_{\lambda(n)})$ /* delegated re-encryption */
35: **if** $n.A_e \cap \mathscr{P}_{\lambda(n)} \neq \emptyset$
36: **then** insert a re-encrypt node $new$ for $n.A_e \cap \mathscr{P}_{\lambda(n)}$ 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** $S \in \mathscr{S}$ **do**
4: $comp\_cost[n,S] = comp\_cost[n_l,S] + comp\_cost[n_r,S] + n.comp\_cost \cdot S.comp\_price$

Figure 1.6: Pseudocode of procedures **Compute_Assignment** and **Compute_Cost**

- *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 **Compute_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 1.7) performs a post-order visit of the query plan to inject encryption and decryption operations as needed. For the root node, the

**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** $n = \mathtt{T}.root$
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** $\mathscr{E}_{\lambda(n_p)} \cap n.vp \neq \emptyset$ **then** insert an encryption node *new* for $\mathscr{E}_{\lambda(n_p)} \cap n.vp$ as parent of $n$
11: $\lambda(new) = \lambda(n)$
12: **Compute_Profile**(*new*); **Compute_Profile**($n$); **Compute_Profile**($n_p$)

Figure 1.7: Pseudocode of procedure **Extend_Plan**

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 6-11). The procedure finally updates the profiles of the nodes impacted by the encryption/decryption operation (line 12).

**Example 1.4.1.** Considering the query plan and authorizations in Figure 1.1, the algorithm first visits the tree in post-order and identifies the candidates for each node (Figure 1.4(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 1.4(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 1.3.1 and operates in $O(|\mathtt{N}| \cdot |\mathscr{S}| \cdot |\mathscr{A}|)$ time, with $\mathscr{A}$ the set of attributes involved in the query.

## 1.5 Summary

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 heuristic aims at limiting the economic cost of query evaluation by choosing, for each node, the candidate that is (locally) more economically convenient.

# 2. Effective and efficient computations over wrapped data

A common concern when using data market services for the management and storage of sensitive data is the threat to the violation of confidentiality of the outsourced data [BDF+19]. Whereas violations of integrity or availability can produce effects that are visible to the customer, violations of confidentiality are usually hard to detect. This is represented by the well-known *honest-but-curious* threat model [SD16]. The classical solution to this threat is represented by the use of encryption, so that the control of the physical representation of the data does not give access to the information content, as long as the external provider does not have access to the encryption key.

A major problem in this context is then enabling the fine-grained access and retrieval of data that are stored in encrypted form in the data market. For real world applications, data are typically structured in relational table, and access requests represented by SQL queries. In the past twenty years, the research and development community have dedicated significant effort to this problem, considering different lines of investigations. Possible approaches include: *i)* the use of searchable encryption (e.g., [PCY+17]), supporting the evaluation of conditions on encrypted data; *ii)* the use of trusted hardware components at the server (e.g., Intel SGX), offering a trusted execution environment residing at, but not accessible by, the server (e.g., [SKLK17]); *iii)* the association with the encrypted data of metadata working as indexes offering support for the evaluation of conditions (e.g., [DDJ+03, HILM02]). All these approaches represent valid alternatives depending on the application scenario. The first two, while enjoying strong protection guarantees, suffer from a significant performance overhead, making them still not applicable in many practical scenarios. On the other hand, indexes, while applicable in practice, may suffer from a possible exposure to inferences, as they might leak information on the values behind them. The vulnerability of indexes typically resides in the frequencies of their occurrences, which can bear relationship with the plaintext values. Frequencies of both individual attributes as well as combinations of them can be exposed to inference. A solution to this problem is guaranteeing indexes with collisions (i.e., mapping different values into the same index), so to provide confusion and indistinguishability. Unfortunately, this is easier said than done, as constructing such an index requires addressing two (interconnected) aspects which are far from being trivial. First, an inevitable curse of dimensionality, while it can easily provide collisions and indistinguishability over one attribute, it is not so when multiple attributes need to be considered. The problem is complicated by the second aspect, which is the need to guarantee effectiveness of indexes (in terms of the limited overhead caused by spurious tuples returned to the clients due to collisions) and their efficiency (in terms of low performance overhead) for query execution. A third nontrivial aspect is the need to limit the storage required at the client for (re)constructing indexes to translate queries on original plaintext data into queries on indexes at the server.

This chapter addresses all these problems and proposes a *multi-dimensional index* that is robust against inference exposure. Such multi-dimensional index ensures not only that index values on

individual attributes are guaranteed to appear at least a given number of times (i.e., no peculiar frequencies can be exploited for inference attacks) but that the same holds for their combination. In other words, each combination of index values enjoys the property of having at least a given number of occurrences. Besides providing protection against static inference attacks (which can no longer exploit frequencies of index values), such an approach guarantees protection against dynamic observations from the storage provider.

The remainder of this chapter is organized as follows. Section 2.1 presents related work and the innovation of MOSAICrOWN. Section 2.2 describes some preliminaries at the basis of the proposed approach. Section 2.3 illustrates the proposed approach for clustering tuples in a multi-dimensional space for indexing. Section 2.4 illustrates the mapping of plaintext values to index values according to the grouping resulting from the clustering. Section 2.5 illustrates the definition of the maps to be maintained at the client-side for query translation. Section 2.6 illustrates the query translation and execution process. Section 2.7 presents the implementation and experimental results confirming the effectiveness and efficiency of our approach as well as the compactness of the maps to be maintained at the client. Section 2.8 concludes the chapter.

## 2.1 State of the art and MOSAICrOWN innovation

We describe the state of the art and the innovation produced by MOSAICrOWN for supporting efficient selective release of data stored in the data market.

### 2.1.1 State of the art

Several research efforts have addressed the problem of supporting queries directly on outsourced encrypted data through the definition of indexing techniques or specific cryptographic schemes (e.g., [DDJ+03, HILM02, PCY+17, PRZB11b, WL06]). Such solutions must be defined with care due to the possible information leakage they may cause (e.g., [CDD+05, NKW15]). The definition of efficient solutions robust against inferences also depends on the specific queries that need to be supported. In particular, the support of range queries requires to define techniques that consider the order relationship characterizing the domain of the attributes on which queries have to be executed, which can complicate the definition of such techniques (e.g., [DDJ+03, DPP+16, VAdE21]). While sharing with our approach the goal of supporting queries over encrypted data, these solutions operate on a single attribute. Our approach instead is based on a multi-dimensional interpretation of the dataset that allows the definition of indexes over multiple attributes. The problem of indexing multi-dimensional datasets has been already considered and resulted in the definition of multi-dimensional indexes for supporting queries with conditions on multiple attributes (e.g., [WHL+14]). These solutions, however, differ from our solution since they define one index only for the whole set of attributes/dimensions considered. Our approach defines a multi-dimensional index with a component for each attribute, considering the intrinsic multi-dimensional nature of relations.

A line of work close to our is represented by approaches aimed at supporting query evaluation over data organized in multiple relations and/or fragments that cannot be joined by unauthorized subjects (e.g., [CSYZ08, DFJ+10, XT06]). These solutions reduce the precision of join operations introducing a degree $k$ of uncertainty (i.e., it is never possible to reconstruct a tuple in the join result with uncertainty lower than $1/k$). Protection is obtained by grouping tuples in the relations/fragments and performing joins at the group (in contrast to tuple) level.

### 2.1.2 MOSAICrOWN innovation

MOSAICrOWN produced several advancements over the state of the art, which can be summarized as follows.

- The definition of a *multi-dimensional index* (i.e., an index on multiple attributes) that is robust against inference exposure and, at the same time, performs well for query execution and requires limited storage at the client-side.
- The proposed multi-dimensional index guarantees protection against dynamic observations from the storage provider, since tuples with the same index values are indistinguishable from one another (so are queries over them). The price to pay for such a protection is the overhead in query execution; being tuples with the same indexes indistinguishable one from the others, any query touching one of them would return all the others as well. Tuples are therefore carefully grouped for indexing so to limit the overhead in query execution and hence guarantee performance. While this can be trivial when only one attribute is to be indexed, it is far from being so (it is an NP-hard problem) when multiple attributes need to be indexed.

The multi-dimensional index approach described in this chapter has been published in [DFF+21b].

## 2.2 Preliminaries

We frame our work in the context of relational database systems (which are still at the basis of almost all applications). We then illustrate our approach with reference to the outsourcing of a relation $r$ defined over schema $R(a_1,\ldots,a_n)$, where each attribute $a_j$ is defined over a domain $\mathtt{d}(a_j)$, for $j=1,\ldots,n$. In the following, we use notation $\mathtt{val}(a_j)$ to denote the set of values of attribute $a_j$ stored in $r$ (i.e., $\mathtt{val}(a_j)$ = SELECT DISTINCT $a_j$ FROM $R$). As an example, Figure 2.1(a) illustrates a relation $r$ with three attributes: `Name`, `State`, and `Age`. Here, $\mathtt{d}(\mathtt{Age})=\{0,\ldots,120\}$ and $\mathtt{val}(\mathtt{Age})=\{27,30,35,38,42,45,50\}$. To protect the confidentiality of data and make them non-intelligible to the storage provider, the owner encrypts the relation before outsourcing it, using a symmetric encryption scheme with a key shared with authorized users only. Queries on the encrypted relation are supported via a set of *indexes* associated with a set $\mathscr{I}=\{a_1,\ldots,a_l\}\subseteq R$ of attributes in the original relation on which conditions need to be evaluated in the execution of queries (`State` and `Age` for our running example). An encrypted and indexed relation is formally defined as follows.

**Definition 2.2.1 (Encrypted and indexed relation).** Let $r$ be a relation over schema $R(a_1,\ldots,a_n)$, and $\mathscr{I}=\{a_1,\ldots,a_l\}\subseteq R$ be a subset of the attributes in $R$. The *encrypted and indexed* version of $r$ is a relation $r^e$ over schema $R^e(et,i_1,\ldots,i_l)$ where $\forall t\in r$, $\exists t^e\in r^e$ such that $t^e[et]{=}E_k(t)$, with $E_k$ a symmetric encryption function with key $k$, and $t^e[i_j]$ the index value derived from $t[a_j]$, $j=1,\ldots,l$.

According to this definition, the encrypted and indexed version $r^e$ of relation $r$ has an attribute *et* that represents the encrypted representation of the tuples in the plaintext relation, and an attribute $i_j$ that represents the index for attribute $a_j$ in $\mathscr{I}$, $j=1,\ldots,l$. Figure 2.1(d) illustrates an example of encrypted and indexed version of the relation in Figure 2.1(a), where `State`, and `Age` are indexed. For simplicity, in the example we use Greek letters to represent index values.

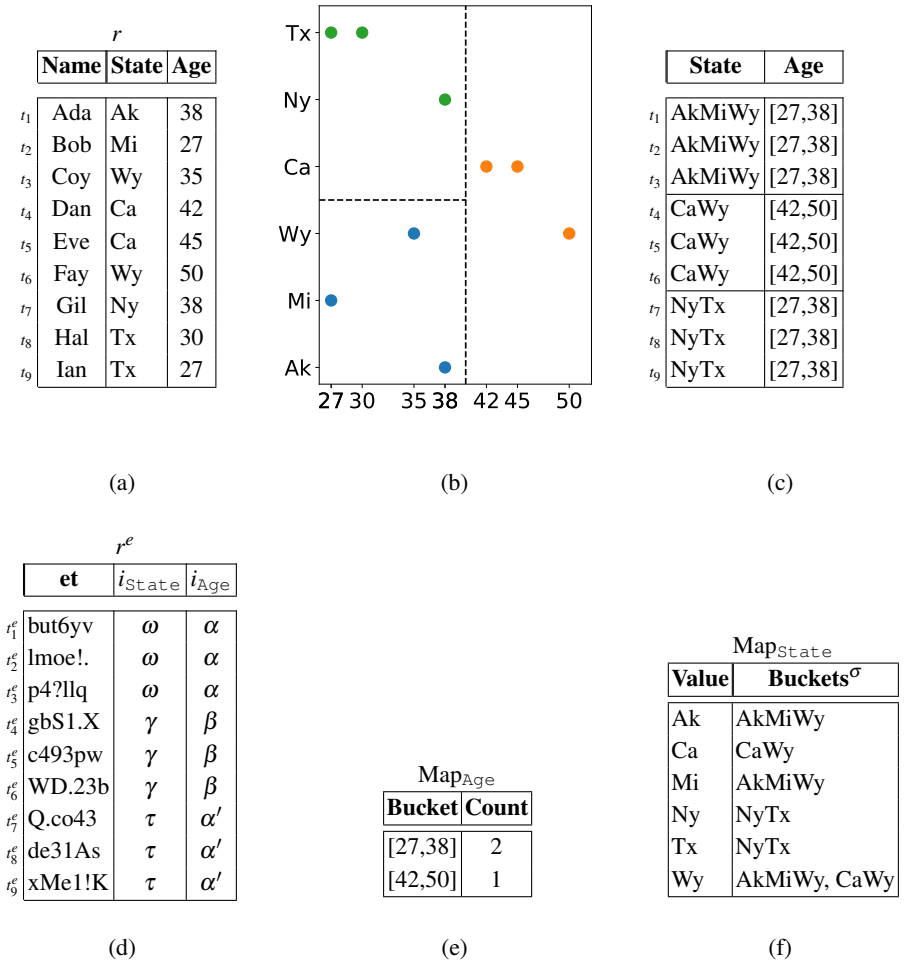



Figure 2.1: Plaintext relation (a), its spatial representation (b), partitioning (c), encrypted and indexed relation (d), and maps for attribute `Age` (e) and `State` (f)

Our goal is to compute a multi-dimensional index that is effective and efficient for the execution of queries with support for equality ($=$) and range ($>, \geq, <, \leq$) conditions.

## 2.3 Multi-dimensional tuple partitioning

Our approach for partitioning tuples for indexing employs an algorithm similar to the one used by the Mondrian anonymization algorithm [DFF+21a, DFF+21c, LDR06]. While similar, our algorithm bears differences to accommodate the fact that we need to cluster tuples to produce obfuscated indexes performing well for query evaluation (in contrast to cluster tuples for semantically meaningful generalization). Our partitioning process works then in a multi-dimensional space, with one dimension for each indexed attribute, and where tuples correspond to points in the multi-dimensional space where their coordinates correspond to the values of the indexed attributes in the tuples. Figure 2.1(b) shows the two-dimensional representation for the indexing of attributes `State` and `Age` of relation in Figure 2.1(a). Since more tuples can have the same values for the indexed attributes, a point in the multi-dimensional space can correspond to more than one tuple,

which is represented in the figure with the number of occurrences associated with it (omitted in our example since it is always equal to 1).

To construct the multi-dimensional space on which the algorithm operates, by partitioning tuples in *boxes* of at least b tuples, we classify attributes to be indexed into two categories:

- *continuous* attributes (e.g., `Age` in Figure 2.1(a)), characterized by a total order relationship on their domain, and on which range conditions need to be supported;
- *nominal* attributes (e.g., `State` in Figure 2.1(a)), which do not have a semantic order in their domain and hence on which only equality conditions make sense.

When partitioning tuples in boxes, care must be taken to put as much as possible tuples with the same values for an attribute in the same box. Also, for continuous attributes, close values should fall as much as possible in the same space. (Note that this might intrinsically not be possible for all attributes.) Consistently with these observations, values of continuous attributes are considered in their natural (we assume increasing) order along the axis of their dimension, while nominal attributes are considered in increasing order of their relative frequencies in the tuples.

The partitioning process works recursively, cutting, at each step, a space (the whole space at the first step) with respect to a selected attribute and a value in its domain as threshold. The cut divides the space in two sub-spaces, each containing the points (i.e., the tuples) falling on its side of the cut. The process is recursively repeated on each of the two resulting sub-spaces, and terminates when any further cut would generate a partition with less than b tuples. At each step, the attribute chosen for the cut is the one that, in the considered space, has the maximum *span*. For continuous attributes, the span is the distance between the minimum and maximum value that the tuples in the (sub-)space assume. For nominal attributes, it is the number of distinct values that the tuples in the (sub-)space assume. For instance, with reference to the relation in Figure 2.1(a), the span for attribute `Age` is $50-27=23$, while the span for attribute `State` is 6. The value chosen as threshold for the cut is the median for continuous attributes, and the value that splits the (sub-)space in two sub-spaces with nearly 50% of the tuples each for nominal attributes. Note that cuts change the relative frequency of values in the generated sub-spaces, and hence also the order in which values for nominal attributes are positioned on their axis.

## 2.4 Index construction

At the end of the partitioning process, the tuples in $r$ are grouped in non-overlapping boxes (i.e., disjoint groups of tuples whose union corresponds to $r$) such that each box contains at least b tuples. The dotted lines in Figure 2.1(b) denote the cuts performed and hence the resulting boxes for our example. Two cuts have been performed, resulting in three boxes, each containing three tuples.

Intuitively, boxes determine the tuples that will be mapped to the same combination of index values. In other words, for each indexed attribute, all the values that fall in the same box will be mapped to the same index. Since this applies to all indexed attributes, this implies that all tuples in the same box will be mapped to the same combination of index values. In the following, we use notation $\mathscr{B}$ to denote the set of all boxes, and $B_j \in \mathscr{B}$ to denote the $j$-th box. Also, for each box $B \in \mathscr{B}$ and attribute $a \in \mathscr{I}$, we denote with $B[a]$ the set of values of $a$, called *bucket*, covered by $B$. A bucket is expressed as an interval for continuous attributes and as a set of values for nominal attributes. Formally, for each box $B$ and attribute $a$:

- $B[a] = [v,v']$ such that $v = \min\{t[a] \mid t \in B\}$ and $v' = \max\{t[a] \mid t \in B\}$, if $a$ is a continuous attribute;
- $B[a] = \{t[a] \mid t \in B\}$, if $a$ is a nominal attribute.

Figure 2.1(c) reports the buckets of the three boxes corresponding to the partitioning in Figure 2.1(b). For readability, we represent each set as a string composed of all its elements, that is, AkMiWy stands for set {Ak,Mi,Wy}.

Note that, different boxes might be associated with the same bucket for one or more of their attributes. Formally, we may have $B_x[a] = B_y[a]$, with $x \neq y$. For instance, in our example, box $B_1$ containing the first three tuples and box $B_3$ containing the last three tuples have $B_1[\texttt{Age}] = B_3[\texttt{Age}] = [27,38]$.

Since we want index values in different boxes to be different, the generation of indexes cannot depend only on the bucket. To map the same bucket of different boxes to different index values, we combine buckets for their indexing with a salt. This is formalized by the following definition.

**Definition 2.4.1 (Index function).** Let $r$ be a relation, $a \in \mathscr{I}$ be an indexed attribute, $\mathscr{B}$ be the set of boxes of relation $r$, and $h_k$ a cryptographic hash function with key $k$. An *index function* for attribute $a$ is a function $\iota_a : \mathscr{B} \rightarrow I_a$ such that:

- $\forall B \in \mathscr{B}$, $\iota_a(B) = h_k(B[a]||\sigma)$, with $\sigma$ a randomly generated salt;
- $\forall a, a' \in \mathscr{I}$, $\forall B, B' \in \mathscr{B}$, with $\iota_a(B) = h_k(B[a]||\sigma)$, $\iota_{a'}(B') = h_k(B'[a']||\sigma')$, if $B[a] = B'[a']$ and $(a \neq a'$ or $B \neq B')$, then $\sigma \neq \sigma'$.

Indexes are then computed as the result of a cryptographic hash function on the concatenation of the bucket to be indexed and a salt. The second bullet in the definition dictates the use of a different salt for buckets that are equal but refer to different attributes (i.e., dimensions) or for a same bucket that appears in different boxes for the same attribute. Satisfaction of such a condition is guaranteed by generating salts using a pseudo-random generation function with a different seed for each attribute. Hence, different attributes will be associated with a different sequence of randomly generated salts. For each attribute $a \in \mathscr{I}$, we denote with $\sigma_a(j)$ the $j$-th salt generated by function $\sigma$ with the seed of attribute $a$. Each bucket $B_x[a]$ is then associated with the $j$-th salt $\sigma_a(j)$, with $j-1$ the number of boxes $B_y \in \mathscr{B}$ such that $B_x[a] = B_y[a]$ and $y < x$. For instance, with reference to the example in Figure 2.1, bucket $B_3[\texttt{Age}]$ is combined with salt $\sigma_{\texttt{Age}}(2)$ since $B_1[\texttt{Age}] = B_3[\texttt{Age}]$ and $1 < 3$. Figure 2.1(d) shows the encrypted and indexed version of the plaintext relation in Figure 2.1(a). Here, combinations of index values $\langle \omega, \alpha \rangle$, $\langle \gamma, \beta \rangle$, and $\langle \tau, \alpha' \rangle$ are those computed for the three boxes in Figure 2.1(c). Indexes $\alpha$ and $\alpha'$ represent different salted versions of the same bucket (i.e., [27,38]).

## 2.5 Client-side maps

The process illustrated in the previous sections enables the creation of indexes to be associated with the tuples in the plaintext relation to be outsourced. The encrypted and indexed relation (Definition 2.2.1) outsourced to the storage provider will then have, for each tuple in the original plaintext relation, its encrypted version and the values of the indexes computed as illustrated (Definition 2.4.1). For simplicity, in our examples we maintain tuples in the same order in the original

and outsourced relations, but clearly tuples should be shuffled before upload them to the storage provider.

The next problem is the definition of the information to be maintained at the client-side to enable the translation of queries on the plaintext relation into queries on the encrypted and indexed relation that can then be executed at the storage provider. According to Definition 2.4.1, such information comprises:

- the cryptographic hash function $h$ along with the corresponding key $k$;
- the function $\sigma_a$ used for salt generation;
- a map, denoted $\text{Map}_a$, enabling the translation of plaintext attribute values into index values.

While the first two bullets only require a client to memorize one function, the third one, requiring attributes' maps, needs more consideration. As maps are being maintained on the client, it is important to maintain them compact, to limit the storage needed at the client. As we will show in Section 2.7, our maps enjoy such compactness. We maintain attributes' maps in a compact form as follows.

**Continuous attributes.** For each continuous attribute $a$, $\text{Map}_a$ is a set of pairs (`Bucket`,`Count`), reporting the buckets in which the attribute has been divided and, for each bucket, the number of boxes in which it appears. Formally, $\text{Map}_a = \{\langle B[a], c\rangle \mid B \in \mathscr{B}, c = |\{B' \in \mathscr{B} \mid B'[a] = B[a]\}|\}$.

The counter associated with each bucket gives the number of distinct index values corresponding to the bucket and hence the number of salts to be used for reconstructing such indexes for query translation. Figure 2.1(e) illustrates the map for attribute `Age` that contains the information about the two buckets resulting from partitioning (i.e., [27,38] and [42,50]). Bucket [27,38] has 2 occurrences and the sequence of salts used in the generation of the corresponding index values is $\sigma_{\texttt{Age}}(1)$ and $\sigma_{\texttt{Age}}(2)$. Hence, the index values corresponding to bucket [27,38] are $h_k([27,38]||\sigma_{\texttt{Age}}(1))=\alpha$ and $h_k([27,38]||\sigma_{\texttt{Age}}(2))=\alpha'$ (see Figure 2.1(d)).

**Nominal attributes.** For each nominal attribute $a$, $\text{Map}_a$ is a set of pairs (`Value`,$\texttt{Buckets}^{\sigma}$) for each actual value $v$ in the actual domain $\texttt{val}(a)$, the buckets $B[a]$ that include $v$, concatenated with the corresponding salt value. Formally, $\text{Map}_a = \{\langle v, \{(B[a] \mid\mid \sigma)\}\rangle \mid v \in \texttt{val}(a), B \in \mathscr{B} : v \in B[a], \iota_a(B) = h_k(B[a] \mid\mid \sigma)\}$.

Figure 2.1(f) illustrates the map for attribute `State`. For simplicity, in the figure, noting that in our example all buckets have only one occurrence, and therefore only one salt is to be used, we report the unsalted bucket values.

## 2.6 Query translation and execution

Once the data have been outsourced, the data (encrypted and indexed) will reside at the external storage provider and only the information needed for query translation (i.e., the maps) will be stored at the client. Each query $q$ on $R$, formulated at the client-side, will then need to be translated on a query $q^s$ operating on indexes at the provider side. The retrieved result (encrypted tuples whose indexes satisfy $q^s$) will be decrypted and query $q^c$ will be executed to eliminate possible spurious tuples, where $q^c$ is the same as $q$ but executed on the decryption of the result of $q^s$ instead of $R$. Figure 2.2 illustrates the overall architecture and query execution process.

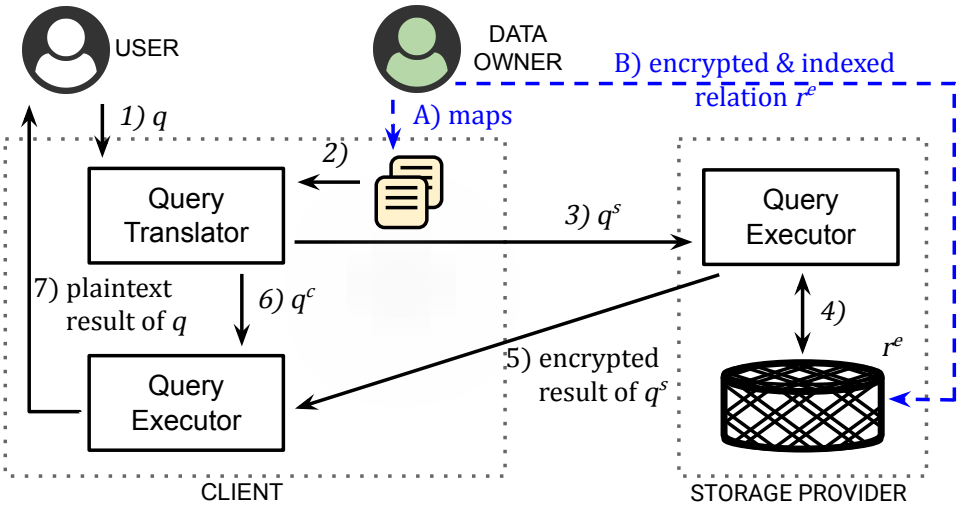



Figure 2.2: Query execution process

Translating $q$ into $q^s$ requires mapping each of the conditions appearing in its WHERE clause into a condition on indexes. We illustrate such a mapping depending on whether the condition is on a continuous or nominal attribute.

**Continuous attribute.** Continuous attributes support both *point* (i.e., equality) as well as *range* conditions. Conditions can then be of the form "$a$ *op* $v$" or "$a$ BETWEEN $v_x$ AND $v_y$", with $a \in \mathscr{I}$, $v, v_x, v_y \in \mathrm{d}(a)$, and *op* $\in \{>, \geq, <, \leq\}$.

To illustrate the mapping, we consider a general form capturing all the cases above and illustrate the mapping of condition "$a \in$ `Range`", where `Range` is an (open or closed) interval specified by two values $v_l$ and $v_r$, with $v_l \leq v_r$. Such a general form captures all the conditions above, by simply considering $v_l = v_r = v$ for point conditions; $v_l$ the lowest value in $\mathtt{val}(a)$ for $<$ or $\leq$ conditions; $v_r$ the highest value in $\mathtt{val}(a)$ for $>$ or $\geq$ conditions; and the interval open on the left (right, resp.) side if values equal to $v_l$ ($v_r$, resp.) should be excluded. For instance, Age$<$30, is equivalent to Age$\in$[27,30).

A condition of the form "$a \in$ `Range`" is translated in a condition on $a$'s index $i_a$, requesting it to be in the set of index values to which the $a$'s buckets intersecting `Range` have been mapped, that is, in condition:

- $i_a$ IN $(h_k(b||\sigma_a(j))$ s.t. $\langle b,c\rangle \in \mathrm{Map}_a$, $b \cap \mathtt{Range} \neq \emptyset$, $j = 1,\ldots,c)$

Here $\langle b,c\rangle \in \mathrm{Map}_a$ denotes the different buckets in $\mathrm{Map}_a$, $b \cap \mathtt{Range} \neq \emptyset$ restricts the consideration to the ones intersecting `Range`, and $c$ expresses the number of salts to be used for each bucket $b$ (i.e., the number of distinct index values to which the bucket has been mapped).

For instance, consider attribute `Age` and its map in Figure 2.1(e). Condition "`Age`=30" is translated as "$i_{\mathtt{Age}}$ IN $(\alpha, \alpha')$", with $\alpha$=$h_k([27,38]||\sigma_{\mathtt{Age}}(1))$ and $\alpha'$=$h_k([27,38]||\sigma_{\mathtt{Age}}(2))$. Condition "`Age`$>$39" is translated as "$i_{\mathtt{Age}}$ IN $(\beta)$", with $\beta$=$h_k([42,50]||\sigma_{\mathtt{Age}}(1))$.

**Nominal attribute.** Nominal attributes support the evaluation of point conditions only. A condition "$a$=$v$" is translated in a condition on $a$'s index $i_a$, requesting it to be in the set of index values to which $v$ has been mapped, that is, in condition:

- $i_a$ IN($\{h_k(set_j)$ such that $m \in \text{Map}_a$, with $m[\texttt{Value}] = v$ and $set_j \in m[\texttt{Buckets}^\sigma]$, $j = 1,\ldots,|m[\texttt{Buckets}^\sigma]|\}$)

For instance, consider attribute `State` and its map in Figure 2.1(f). Condition "`State`=Wy" is translated as "$i_{\texttt{State}}$ IN ($\omega$, $\gamma$)", with $\omega$=$h_k$(AkMiWy) and $\gamma$=$h_k$(CaWy). Condition "`State`=Ca" is translated into condition "$i_{\texttt{State}}$ IN ($\gamma$)".

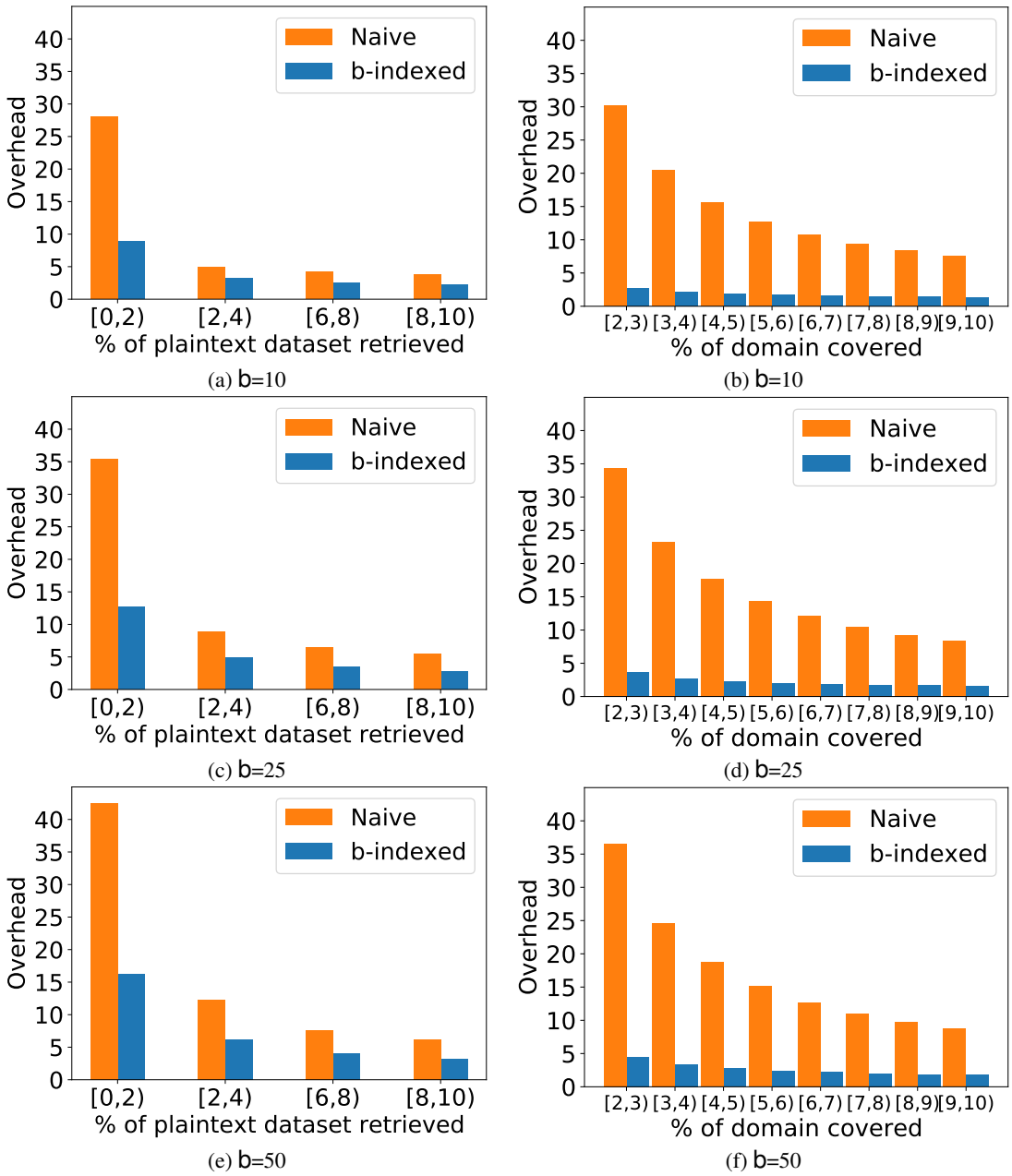
## 2.7 Implementation and experiments

We built a prototype in Python that implements both the generation of the encrypted and indexed relation and the query evaluation process. We then tested our approach on the PUMS USA ACS 2019 dataset, containing 3.2M tuples [RFG+20]. The dataset schema includes two nominal (`State` and `Occupation`) and two continuous (`Age` and `Income`) attributes. Our experiments analyze the overhead in query execution and the size of the client-side maps.

**Indexing and encryption.** We realized a distributed version of the partitioning process illustrated in Section 2.3. Our tool uses the scalable Apache Spark platform and parallelizes the partitioning process relying on an arbitrary number of workers. The dataset is partitioned to the workers and they independently apply the partitioning process on the tuples assigned to them. Each worker computes index values (Definition 2.4.1) using the Blake2b hash function and a different 16-byte salt for each attribute. The encrypted tuple (attribute *et*) is computed using XSalsa20 with Poly1305 MAC (to guarantee both confidentiality and integrity) with a 32-byte high-entropy key. The tool also generates a fresh nonce for each encryption invocation to provide indistinguishability of tuples with the same values for all encrypted attributes. The encryption functions are implemented by *PyNacl*. The encrypted and indexed relations generated by each worker are uploaded, in randomized order, to a containerized PostgreSQL DBMS.

**Query translation.** Each client-side query $q$ is parsed using *sqlparse*, a SQL parser for Python, and translated as described in Section 2.6. Query $q^s$ is submitted to the PostgreSQL DBMS hosted at the storage provider, and its result is decrypted and checked for integrity by the client. The client executes query $q^c$ on an in-memory SQLite DB to filter spurious tuples and project the attributes of interest.

**Query overhead.** We compared our solution with a Naive approach that builds boxes of $\mathsf{b}$ tuples by ordering tuples according to the values of a sequence of attributes and then splitting the ordered dataset in boxes of $\mathsf{b}$ contiguous tuples. We run two kinds of query: *1)* point queries for each attribute $a$ in the dataset schema (i.e., `State`, `Occupation`, `Age`, `Income`), and each value $v$ in `val`($a$); *2)* range queries for attribute `Age` and for each range $[v_i,v_j]$ of values in `val`(`Age`). Figure 2.3 compares the overhead in query execution for the Naive approach and for our $\mathsf{b}$-indexed approach. Figures 2.3(a,c,e) refer to point queries and Figures 2.3(b,d,f) refer to range queries. The overhead is measured in terms of the average ratio between the number of tuples returned by the query on index $i_a$ and the original query on $a$, considering $\mathsf{b}$ with values 10, 25, and 50. For point queries, such an overhead is measured depending on the selectivity of the original query on the plaintext data (plotted on the $x$ axis and expressed in terms of percentage of tuples returned). We assumed each query to be issued as many times as the frequency of the requested value. For range queries, Figures 2.3(b,d,f), the overhead is measured depending on the percentage of domain values covered by the range condition (plotted on the $x$ axis). As visible from the figures, our approach largely outperforms the Naive one, whose overhead compared to ours is between 1.5x and 3x for point queries and between 5x and 10x for range queries.




(a) b=10 (b) b=10

(c) b=25 (d) b=25

(e) b=50 (f) b=50

Figure 2.3: Point (a,c,e) and range (b,d,f) queries overhead

It is interesting to note how the limited overhead provided by our b-indexed approach, which is much smaller than the value of b. Such limited overhead is to be particularly appreciated especially in comparison with alternative for privacy-aware query execution that suffers at least 30x overhead in size of each access (in some cases a more than 1000x overhead), which grows significantly when indexes are introduced.

**Local data structure.** Figure 2.4(a) illustrates the size of the maps stored at the client-side varying the number of tuples in the dataset between 0.5 and 3 millions and b equal to 10, 25, and 50. As visible from Figure 2.4(b), our maps require in almost all the analyzed configurations less than 1 byte per tuple. It is to note that the storage space per tuple required by the map decreases as the number of tuples increases.

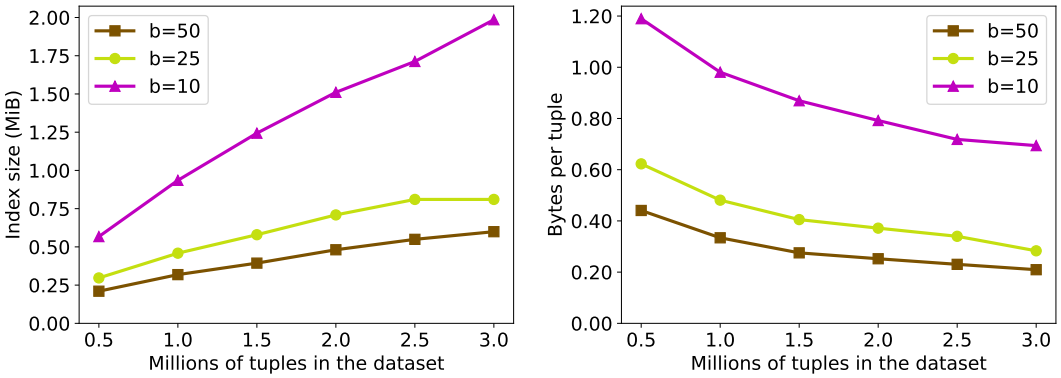



Figure 2.4: Absolute (a) and relative (b) size of the maps

## 2.8 Summary

The chapter described an approach for addressing the problem of storing encrypted data in the data market and defining indexes over them for enabling query execution. The proposed approach to index construction, working on multiple attributes guarantees protection against inferences, while providing effective and efficient query execution, with support for both point and range conditions. The experimental evaluation on large publicly available datasets confirms the validity of our approach and therefore its applicability in practical scenarios.

# 3. Conclusions

In this deliverable, we have presented the advancements of the work carried out in MOSAICrOWN Work Package 4. WP4 has addressed the problem of defining efficient techniques aimed to wrap data with a protection layer (typically removable), while ensuring access functionality without compromising on protection. The deliverable has then focused on techniques for supporting controlled data sharing in collaborative queries and fine-grained data retrieval.

Chapter 1 has illustrated an approach for the management of collaborative queries when the involved datasets are associated with access restrictions and the subjects involved in query execution have partial access to the data. Datasets can be stored either in plaintext or encrypted form. Authorizations associated with datasets are defined by their authorities, and regulate access to data specifying if a data item can be accessed by a subject in plaintext or in encrypted form. Enforcement of authorizations considers the possibility that data stored in encrypted form need to be decrypted and then re-encrypted to support the execution of specific operations over them. The chapter has also described a heuristic for determining a minimum cost query plan. The proposed heuristic algorithm determines an assignment of operations to subjects in complete obedience of authorizations and such that for each query operation, the subject in charge of its execution is the subject (locally) more economically convenient. MOSAICrOWN is working on an enhancement of this proposal to support trusted hardware components in query execution.

Chapter 2 has addressed the problem of outsourcing encrypted data to the digital data market and defining indexes over them for enabling fine-grained data retrieval. The proposed approach for index construction works on multiple attributes and guarantees protection against inferences. The idea is to partition the tuples in a dataset into boxes of a given size and then to associate all tuples in the same box with the same combination of index values, ensuring that different boxes have different combinations of index values. All tuples in a box are always returned together in the result of a query, making them indistinguishable from all the tuples in the same box. This implies that boxes must be carefully constructed to limit the overhead in query execution and hence guarantee performance. The experimental evaluation on large publicly available datasets confirms the validity of the proposed approach and therefore its applicability in practical scenarios. MOSAICrOWN is working on an enhancement of this proposal to support a perfect flat indexing where a dataset is stored in groups of exactly the same number of data items. MOSAICrOWN is also investigating the use of a key-value store for the storage of datasets in contrast to the PostgreSQL DBMS used in the approach described in this deliverable.

# Bibliography


[AAC+18] W. Alkowaileet, S. Alsubaiee, M.J. Carey, C. Li, H. Ramampiaro, P. Sinthong, and X. Wang. End-to-end machine learning with apache asterixdb. In *Proc. of DEEM*, Houston, TX, USA, June 2018.

[AAKL06] R. Agrawal, D. Asonov, M. Kantarcioglu, and Y. Li. Sovereign joins. In *Proc. of ICDE*, Atlanta, GA, April 2006.

[AB18] A. Amarilli and M. Benedikt. When can we answer queries using result-bounded data interfaces? In *Proc. of PODS*, Houston, TX, USA, June 2018.

[AXL+15] M. Armbrust, R. Xin, C. Lian, Y. Huai, D. Liu, J.K. Bradley, X. Meng, T. Kaftan, T. Kaftam, M.J. Franklin, A. Ghodsi, and M.A. Zaharia. Spark SQL: Relational data processing in Spark. In *Proc. of SIGMOD*, Melbourne, Australia, May-June 2015.

[BDF+19] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati. Dynamic allocation for resource protection in decentralized cloud storage. In *Proc. of GLOBECOM*, Waikoloa, Hawaii, USA, December 2019.

[BEE+17] 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, February 2017.

[BLT15] M. Benedikt, J. Leblay, and E. Tsamoura. Querying with access patterns and integrity constraints. *PVLDB*, 8(6):690–701, February 2015.

[CDD+05] A. Ceselli, E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM TISSEC*, 8(1):119–152, February 2005.

[CLS09] S. Chow, J.-H. Lee, and L. Subramanian. Two-party computation model for privacy-preserving queries over distributed databases. In *Proc. of NDSS*, San Diego, CA, USA, February 2009.

[CSYZ08] G. Cormode, D. Srivastava, T. Yu, and Q. Zhang. Anonymizing bipartite graph data using safe groupings. *PVLDB*, 1(1):833–844, August 2008.

[DCL19] E.B. Dimitrova, P.K. Chrysanthis, and A.J. Lee. Authorization-aware optimization for multi-provider queries. In *Proc. of SAC*, Limassol, Cyprus, April 2019.

[DDJ+03] E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *Proc. of ACM CCS*, Washington, DC, USA, October 2003.

[DFF+21a] S. De Capitani di Vimercati, D. Facchinetti, S. Foresti, G. Oldani, S. Paraboschi, M. Rossi, and P. Samarati. Artifact: Scalable distributed data anonymization. In *Proc. of PerCom*, Kassel, Germany (virtual), March 2021.

[DFF+21b] S. De Capitani di Vimercati, D. Facchinetti, S. Foresti, G. Oldani, S. Paraboschi, M. Rossi, and P. Samarati. Multi-dimensional indexes for point and range queries on outsourced encrypted data. In *Proc. of the GLOBECOM*, Madrid, Spain, December 2021.

[DFF+21c] S. De Capitani di Vimercati, D. Facchinetti, S. Foresti, G. Oldani, S. Paraboschi, M. Rossi, and P. Samarati. Scalable distributed data anonymization. In *Proc. of PerCom*, Kassel, Germany (virtual), March 2021.

[DFJ+10] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragments and loose associations: Respecting privacy in data publishing. *PVLDB*, 3(1):1370–1381, September 2010.

[DFJ+11] 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.

[DFJ+14] 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, November/December 2014.

[DFJ+17] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati. An authorization model for multi-provider queries. *PVLDB*, 11(3):256–268, November 2017.

[DFJ+21] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati. Distributed query evaluation over encrypted data. In *Proc. of DBSec*, Calgary, Canada (virtual), July 2021.

[DPP+16] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis. Practical private range search revisited. In *Proc. of ACM SIGMOD*, San Francisco, CA, USA, June–July 2016.

[FL20] S. Foresti and G. Livraga. D4.1 – First Version of Encryption-based Protection Tools. Technical report, MOSAICrOWN, May 2020.

[FLCY14] N.L. Farnan, A.J. Lee, P.K. Chrysanthis, and T. Yu. PAQO: Preference-aware query optimization for decentralized database systems. In *Proc. of ICDE*, Chicago, IL, USA, March–April 2014.

[FP20] S. Foresti and S. Paraboschi. D4.2 – Report on Encryption-Based Techniques and Policy Enforcement. Technical report, MOSAICrOWN, June 2020.

[GB14] M. Guarnieri and D. Basin. Optimal security-aware query processing. *PVLDB*, 7(12):1307–1318, August 2014.

[HILM02] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of ACM SIGMOD*, Madison, Wisconsin, USA, June 2002.

[HIML02] H. Hacigümüs, B. Iyer, S. Mehrotra, and C. Li. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of SIGMOD*, Madison, WI, USA, June 2002.

[KB16] M.M. Kwakye and K. Barker. Privacy-preservation in the integration and querying of multidimensional data models. In *Proc of PST*, Auckland, New Zealand, December 2016.

[Kos00] D. Kossmann. The state of the art in distributed query processing. *ACM CSUR*, 32(4):422–469, December 2000.

[LDR06] K. LeFevre, D.J. DeWitt, and R. Ramakrishnan. Mondrian multidimensional *k*-anonymity. In *Proc. of ICDE*, Atlanta, GA, USA, April 2006.

[LSK95] A.Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *JIIS*, 5(2):121–143, September 1995.

[NKW15] M. Naveed, S. Kamara, and C.V. Wright. Inference attacks on property-preserving encrypted database. In *Proc. of CCS*, Denver, Colorado, USA, October 2015.

[OKM17] K.Y. Oktay, M. Kantarcioglu, and S. Mehrotra. Secure and efficient query processing over hybrid clouds. In *Proc. of ICDE*, San Diego, CA, USA, April 2017.

[PCY+17] G.S. Poh, J. Chin, W. Yau, K.R. Choo, and M.S. Mohamad. Searchable symmetric encryption: Designs and challenges. *ACM CSUR*, 50(3):40:1–40:37, May 2017.

[PRZB11a] R.A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *Proc. of SOSP*, Cascais, Portugal, October 2011.

[PRZB11b] R.A. Popa, C.M.S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *Proc. of SOSP*, Cascais, Portugal, October 2011.

[RFG+20] S. Ruggles, S. Flood, R. Goeken, J. Grover, E. Meyer, J. Pacas, and M. Sobek. IPUMS USA: Version 10.0 [dataset], 2020. https://doi.org/10.18128/D010.V10.0.

[RLG17] A. Rheinländer, U. Leser, and G. Graefe. Optimization of complex dataflows with user-defined functions. *ACM CSUR*, 50(3):38:1–38:39, October 2017.

[RMSR04] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proc. of SIGMOD*, Paris, France, June 2004.

[SBM20] S. Sharma, A. Burtsev, and S. Mehrotra. Advances in cryptography and secure hardware for data outsourcing. In *IEEE ICDE*, Dallas, TX, USA, April 2020.

[SD16] P. Samarati and S. De Capitani di Vimercati. Cloud security: Issues and concerns. In S. Murugesan and I. Bojanova, editors, *Encyclopedia on Cloud Computing*. Wiley, 2016.

[SKLK17] F. Shaon, M. Kantarcioglu, Z. Lin, and L. Khan. SGX-BigMatrix: A practical encrypted data analytic framework with trusted processors. In *Proc. of ACM CCS*, Dallas, TX, USA, October-November 2017.

[SKS+19] G. Salvaneschi, M. Köhler, D. Sokolowski, P. Haller, S. Erdweg, and M. Mezini. Language-integrated privacy-aware distributed queries. *Proc. of ACM Programming Language*, 3:167:1–167:30, October 2019.

[TKMZ13] S. Tu, M.F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. *PVLDB*, 6(5):289–300, March 2013.

[VAdE21] H. Van Tran, T. Allard, L. d'Orazio, and A. El Abbadi. FRESQUE: A scalable ingestion framework for secure range query processing on clouds. In *Proc. of EDBT*, Nicosia, Cyprus, March 2021.

[WHL+14] B. Wang, Y. Hou, M. Li, H. Wang, and H. Li. Maple: Scalable multi-dimensional range search over encrypted cloud data with tree-based index. In *Proc. of ACM ASIACCS*, Kyoto, Japan, June 2014.

[WL06] H. Wang and L.V.S. Lakshmanan. Efficient secure query evaluation over encrypted XML databases. In *Proc. of VLDB*, Seoul, Korea, September 2006.

[XT06] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *Proc. of VLDB*, Seoul, Korea, September 2006.

[ZZL+15] 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, April 2015.