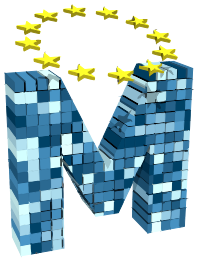


Project title: Multi-Owner data Sharing for Analytics and Integration respecting Confidentiality and OWNer control
Project acronym: MOSAICrOWN
Funding scheme: H2020-ICT-2018-2
Topic: ICT-13-2018-2019
Project duration: January 2019 – December 2021

D4.2

Report on Encryption-Based Techniques and Policy Enforcement

Editors: Sara Foresti (UNIMI)
Stefano Paraboschi (UNIBG)
Reviewers: Jonas Böhler (SAP SE)
Aidan O Mahony (EISI)

Abstract

This deliverable describes the advanced encryption-based techniques developed in MOSAICrOWN for enforcing data protection in digital data markets. These techniques enable wrapping data with an encryption-based layer of protection when they are stored, accessed, and processed in the data market. The techniques described in this document focus, in particular, on: 1) the controlled, on-the-fly and selective, application of encryption to protect confidential information in data processing and computation, and 2) the application of an All-Or-Nothing Transform encryption mode, combined with fragmentation and slicing, for protecting data in storage, while ensuring resilience to failures, and strong protection even in case of key leakage, and 3) the owner-controlled wrapping of data when ingested in the digital market for enriching data with a self-protecting layer to enforce authorizations.

Type	Identifier	Dissemination	Date
Deliverable	D4.2	Public	2020.06.30

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

Versions

Version	Date	Description
0.1	2020.06.04	Initial Release
0.2	2020.06.25	Second Release
1.0	2020.06.30	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	Stefano Paraboschi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 1: 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: AONT with splitting and fragmentation	Enrico Bacis (UNIBG), Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Stefano Paraboschi (UNIBG), Marco Rosa (UNIBG), Pierangela Samarati (UNIMI)
Chapter 3: Data wrapping for controlled sharing	Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Giovanni Livraga (UNIMI), Pierangela Samarati (UNIMI)
Chapter 4: Conclusions	Stefano Paraboschi (UNIBG), Pierangela Samarati (UNIMI)

Contents

List of Figures

Executive Summary

This document describes the advanced encryption-based solutions developed in MOSAICrOWN for protecting confidentiality of data ingested, stored, and processed in a digital data market. It addresses different aspects of the problem, to ensure proper data protection while guaranteeing access and processing functionality in the different phases of the data life cycle in the digital data market (i.e., ingestion, storage, and processing). Leveraging such protection also enables an enriched functionality of the data market that can see the involvement, for data storage, access, and computation, of parties not fully trusted or not fully authorized for complete data access. The document is organized in three chapters.

Chapter 1 presents our techniques to enable the protection of data in the context of data analytics and processing, and – in particular – generic query execution. Our approach supports three levels of visibility for parties over the data: plaintext visibility, encrypted visibility, and no visibility. It captures and controls the data flows enacted by the execution of a query plan and computes an optimal assignment of the different steps of the computation to different parties, based on their authorizations as well as their cost. The aim is the identification of an assignment for the execution of the operations in the query plan that is both economically convenient and compliant with authorizations. Our solution enriches the query plan execution with the application of encryption as needed to ensure that parties involved in the computation will be able to (directly or indirectly) view only the data they are authorized to access. Encryption will then be injected on-the-fly in the query plan execution, dynamically enforcing data wrapping and unwrapping.

Chapter 2 addresses the protection of data ingested and stored in the data market. It presents the adoption of an advanced solution providing All-Or-Nothing Transform (AONT) encryption, which ensures that data remain protected in their entirety even when the data market (or part of it) is not fully trusted or when the encryption key is leaked or compromised. AONT encryption enables resource deletion and the dynamic enforcement of access revocation without requiring complete re-encryption of possibly large resources. Our approach enriches the application of AONT with a solution providing for slicing resources so to distribute their, possibly replicated, allocation and storage at different parties considering scenarios of data markets leveraging decentralized cloud storage services. Our modeling allows the identification of confidentiality and availability guarantees provided by the combination of slicing and replication parameters, hence enabling the data owner to set them in such a way to best suit the needs of each specific scenario.

Chapter 3 presents the design of a solution leveraging encryption for enabling data owners to wrap data with a self-protecting layer when ingesting them in the data market. The approach is based on the use of hierarchical encryption and key derivation to provide efficient key management. It enables owners to ingest an encrypted/wrapped version of their data in the data market and enables consumers to access them by properly demonstrating their ability to derive the data decryption key. Our solution also puts forward the idea of combining encryption and key management with blockchain and smart contract technologies, towards the realization of a data market empowering owners to realize economic incentives when making their data available to others.

1. Dynamic wrapping for collaborative computations

In this chapter, we discuss the problem of enabling efficient collaborative evaluation of computations over data managed by the cloud market, owned by different parties. The data market represents a promising solution for combining data from different sources, to the aim of extracting useful information. Since computation of large data collections can be expensive, we take into consideration the possibility for the data market to participate and possibly partially delegate query evaluation to third parties, whenever it provides an economic advantage while not violating the need for protection of confidential information by data owners. Clearly, the release of information by data owners for collaborative computations needs to be regulated according to the owners' requirements, making data release selective.

In this chapter, we illustrate a solution aimed at enabling the involvement of external third parties in query evaluation, while guaranteeing the satisfaction of the authorization policy regulating data release defined by data owners. The approach illustrated in this chapter builds on the proposal in [DFJ+17], enabling the support of a larger number of operators and properly managing (attribute) rename operations.

The remainder of this chapter is organized as follows. Section 1.1 presents related works and the innovation of MOSAICrOWN. Section 1.2 introduces the preliminary results on which our solution builds. Section 1.3 illustrates the problems related with authorization enforcement when attributes are renamed. Section 1.4 defines the relation profile resulting from the evaluation of rename and set operations, and of arithmetic expressions. Section 1.5 discusses authorization enforcement, and the computation of the assignment of operations in a query plan that minimizes query evaluation costs, while satisfying authorizations. Section 1.6 concludes the chapter.

1.1 State of the art and MOSAICrOWN innovation

In this section, we illustrate the state of the art and the innovation produced by MOSAICrOWN for collaborative computations over data stored in the market, possibly owned by different authorities.

1.1.1 State of the art

The problem of managing queries in distributed scenarios has been extensively studied, but traditional solutions (e.g., [Kos00, LSK95]) as well as modern approaches that consider big data analytics (e.g., [AAC+18, AXL+15, RLG17]) do not take into consideration access restrictions. In the relational database context, access restrictions can be supported by views (e.g., [DFJ+14, GB14, RMSR04]), access patterns (e.g., [AB18, BLT15]), or data masking (e.g., [KB16]). Such proposals however do not consider encryption.

Different works have addressed the problem of protecting data confidentiality in distributed computations (e.g., [DFJ+11, OKM17, SKS+19, ZZL+15]). In [ZZL+15] the authors present an approach to collaboratively execute queries on data subject to access restrictions, considering different join evaluation strategies. In [SKS+19] the authors propose an operator placement approach aimed at satisfying privacy constraint, while maximizing performance in query evaluation. The proposed solution relies on programming language techniques for regulating and controlling information flows. In [DFJ+11] the authors provide a solution for restricting access and sharing of distributed data, which supports the explicit consideration of join paths in the authorizations. The proposal in [OKM17] aims to protect MapReduce computations in hybrid clouds, preventing flows of sensitive information to the public cloud. These works confirm the relevance of the problem but focus on different aspects. None of the proposals considers the possibility of protecting data with encryption. The idea of specifying different visibility levels over data has been first proposed in [DFJ+17]. The approach in [DCL19] integrated this authorization model in a distributed query optimizer.

Several works (e.g., [AAKL06, HIML02, PRZB11, TKMZ13]) have investigated the use and support of encryption for the protection of data in storage or query execution. Other approaches (e.g., [BEE+17, CLS09]) proposed solutions for using secure multiparty computation in query evaluation, to keep both the input operands and the result secret to the party in charge of query evaluation.

1.1.2 MOSAICrOWN innovation

MOSAICrOWN produced several advancements over the state of the art, which are discussed in this section.

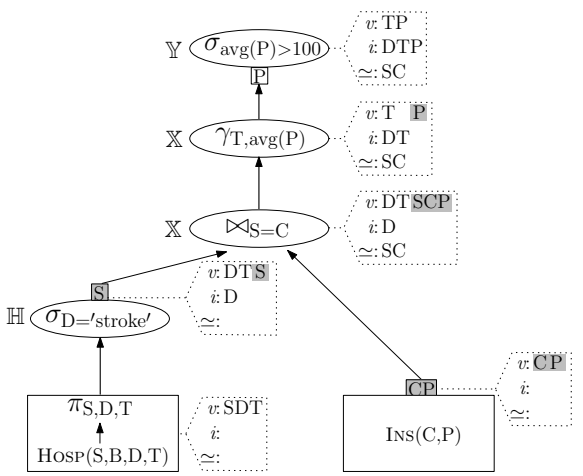
- The first innovation is represented by the analysis and support of additional operators (i.e., rename operation, set operations, and arithmetic expressions) with respect to the ones considered in the original proposal.
- The second innovation consists of the definition of an extended relation profile, which permits to keep track of derived attributes (i.e., attributes resulting from a rename operation or from the evaluation of an expression). The verification of releases is therefore revised and extended to take into account such an extended relation profile to manage attribute names that have not been used in the definition of authorizations.

Some of the results obtained by MOSAICrOWN and illustrated in this chapter have been published in [BDF+19a]. A preliminary version of the proposal illustrated in this chapter has been implemented by one of the tools presented in deliverable D4.1 “First version of encryption-based protection tools” [FL20].

1.2 Preliminaries

For concreteness, we frame our work in the context of the execution of queries over relational tables, performed according to a query plan represented as a tree whose leaves are base relations and whose non-leaf nodes are operations to be executed to perform the query.

The two building blocks of the approach illustrated in this chapter are the authorization model regulating data visibility, and the definition of relation profiles capturing the informative content (directly or indirectly) conveyed by relations.



(a) (b)

Figure 1.1: An example of a query plan enriched with relation profiles, assignees for the execution of its operations, and encryption/decryption operations (a), and of a set of authorizations (b)

Authorization model. According to the authorization model proposed in [DFJ+17], authorizations specify whether a subject can have, for an attribute:

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

Authorizations are then defined as follows.

Definition 1.2.1 (Authorization) *Let R be a relation and $\mathcal{S}$ be a set of subjects. An* authorization *is a rule of the form $[P,E]\rightarrow S$, where $P\subseteq R$ and $E\subseteq R$ are subsets of attributes in R such that $P\cap E=\emptyset$, and $S\in\mathcal{S}\cup\{any\}$.*

Assuming a closed policy, a subject can hold at most one authorization for each relation, stating which attributes (P) she can view in plaintext and which attributes (E) she can view encrypted. The subject cannot view any other attribute in the relation schema. Clearly, if S can access an attribute a in plaintext she can also access its encrypted version. The consideration of encrypted visibility enables subjects who are not trusted to access the sensitive data content to perform computations over them in encrypted form.

Figure 1.1(b) illustrates a set of authorizations over relations Hospital(SSN, BirthDate, Disease, Treatment) and Insurance(Customer, Premium) for a set of six subjects $\mathbb{H}$ (owner of relation Hospital), $\mathbb{I}$ (owner of relation Insurance), $\mathbb{U}$ (the user posing the query), $\mathbb{X}$, $\mathbb{Y}$, and $\mathbb{Z}$ (subjects offering computational capabilities), along with the default authorization for 'any'. In the figure, attributes are denoted by their initials and, for the sake of readability, in the authorizations we denote a set of attributes simply with the sequence of the attributes composing it, omitting the curly brackets and commas (e.g., SBDT stands for {S,B,D,T}).

Relation profiles. A relation resulting from a computation can convey information on attributes not explicitly appearing in its schema. The profile of a relation includes: visible attributes that appear in the relation schema (in plaintext or encrypted), implicit attributes taken into account in the computation of the relation (in plaintext or encrypted), and equivalence relationships among attributes compared in the computation. Intuitively, implicit attributes are all those attributes that appear in a selection condition or grouping operation in the (sub-)query producing the relation. For instance, attribute B is implicit in the profile of the relation resulting from the evaluation of query "SELECT A FROM R WHERE B='10'". Attributes are considered equivalent if they have been compared in a computation and therefore visibility of one attribute indirectly leaks the other(s). For instance, attributes A and B are equivalent in the profile of the relation resulting from the evaluation of query "SELECT A FROM R_1 JOIN R_2 ON A=B".

The profile of the relation resulting from the evaluation of an operation depends on the operator and on the profile of its operands. Figure 1.2 illustrates the profile resulting from the evaluation of: projection, selection, cartesian product, join, and group by operators. For each operator, the figure reports the general formula on the left and a simple example on the right. In the figure, the profile of a relation is represented as a tag attached to the node generating it, where v denotes the visible component, i the implicit component, and $\simeq$ the equivalence relationships. Encrypted attributes are represented on a gray background. The figure also reports encryption (gray box on top of the operand relation) and decryption (white box below the subsequent operator) operations.

Since profiles capture the informative content of relations, they can be used to verify the satisfaction of authorizations when releasing a relation, authorizing a subject for a relation based on whether she can legitimately access all the informative content carried by a relation, and for the execution of an operation in a query plan if she is authorized for the input and output relations. Given a query plan, the solution illustrated in [DFJ+17] injects encryption and decryption operations to enable different subjects to execute different operations, in full respect of the authorizations.

1.3 Derived attributes

The preliminary model illustrated in Section 1.2 supports the execution of computations involving projections, selections, cartesian products, joins, and group by operations. Computations involving other operations, such as set operations (union $\cup$, intersection $\cap$, and difference $\setminus$) and arithmetic operations, could not be handled. The consideration of such additional operations introduces new challenges that need to be carefully addressed, mainly due to the possibility of introducing, during the computation, *derived attributes*. Derived attributes are attributes not appearing in any original relation schema and obtained through either a renaming operation or an arithmetic/set operation, with new names dictated by the computation itself. The release of a relation with derived attributes clearly discloses them, even if they do not appear in the original relation schemas. While such derived attributes might resemble the concept of implicit attributes investigated in the preliminary model, unfortunately they cannot be directly managed as such. In particular, the information leakage due to derived attributes complicates the enforcement of the authorization policy, since authorizations are defined over the original attributes appearing in the relation schemas and do not regulate the release of attributes with different (new) names. For instance, query "SELECT A AS B FROM R" reveals the values of attribute A under name B, but no authorization regulates the release of B since B does not belong to the original schema of R. Clearly, the authorizations originally defined over A must apply also to B, since A and B are two different names for the same attribute.

In MOSAICrOWN, we have investigated this issue and proposed a solution for correctly man-

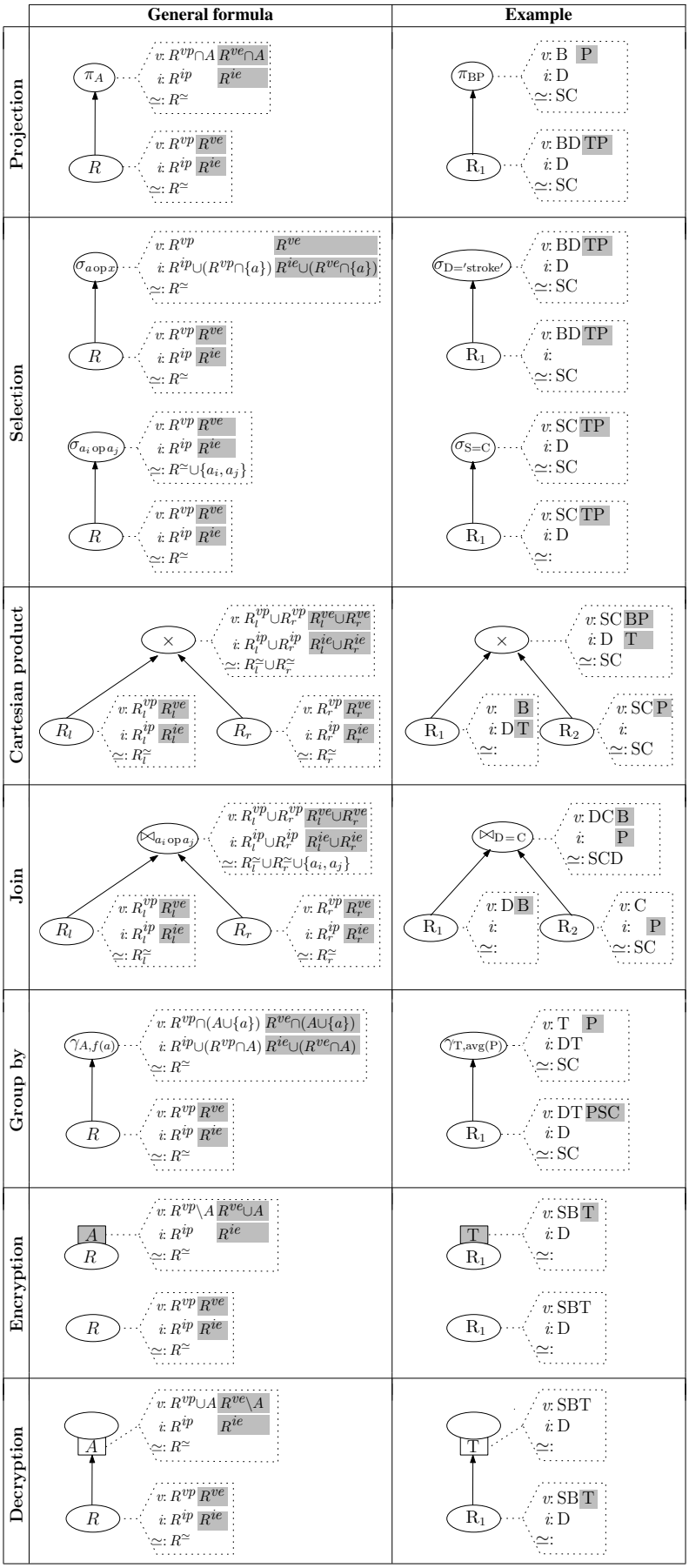


Figure 1.2: Graphical representation of the profiles resulting from relational and encryption/decryption operations

aging derived attributes. This leads to a more complete model that allows for more complex operations involving, besides basic relational operators, also advanced operators such as set operators and arithmetic expressions. In the remainder of this chapter, we illustrate how derived attributes can be captured in relation profiles, how they can be impacted by the operations involved in a query, and how the authorizations can then be enforced when assigning query operations to subjects for their execution.

1.4 Relation profile

We define the *profile* of a relation to capture the informative content carried by the relation in terms of attributes explicitly as well as implicitly visible and taking into account information conveyed by equivalent and derived attributes. We refer to attributes explicitly visible in a relation as *visible* attributes, and to those implicitly leaked as *implicit*. In addition, attributes can be represented in *plaintext* or *encrypted*. In the following, we will refer to non-derived attributes (i.e., attributes appearing in the original relation schemas) as *base attributes*.

Definition 1.4.1 (Relation Profile) *The* profile *of a relation R is a 6-tuple of the form $[R^{vp}, R^{ve}, R^{ip}, R^{ie}, R^{\simeq}, R^{\rightarrow}]$ where: R^{vp} and R^{ve} are the* visible *attributes appearing in R's schema in plaintext (R^{vp}) or encrypted (R^{ve}) form; R^{ip} and R^{ie} are the* implicit *attributes conveyed by R, in plaintext (R^{ip}) or encrypted (R^{ie}) form; $R^{\simeq}$ is a disjoint-set data structure representing the closure of the equivalence relationship implied by attributes connected in R's computation; and $R^{\rightarrow}$ is a set of pairs $[a,A]$ expressing the fact that attribute a has been derived from the set A of base attributes.*

Intuitively, the profile of a relation is extended to keep track of the correspondence between the attributes derived from a computation (i.e., an aggregate function or an arithmetic expression) and the attributes from which they have been obtained. Such a correspondence is needed to verify whether these new attributes can be released to a given subject as this depends on the base attributes (which are the only attributes explicitly mentioned in the authorizations) from which they have been computed.

Even if a new attribute a can be derived from both base attributes and derived attributes, we can easily look at $R^{\rightarrow}$ component as composed of set of pairs $[a,A]$ where A includes base attributes only. Indeed, the $R^{\rightarrow}$ component of a relation profile can be easily used to translate a pair $[a,A]$ into a new pair $[a,A']$ where A' includes only base attributes. For instance, assume that $R^{\rightarrow}$=$\{[a_3,\{a_1,a_2\}]\}$ and that attribute a_5 derives from attributes a_3 and a_4, meaning that the pair $[a_5,\{a_3,a_4\}]$ must be added to $R^{\rightarrow}$. In this case, since attribute a_3 derives from $\{a_1,a_2\}$, then, by transitivity, a_5 derives from $\{a_1,a_2,a_4\}$, and pair $[a_5, \{a_1,a_2,a_3\}]$ can then be added to $R^{\rightarrow}$. Formally, given an attribute a and component $R^{\rightarrow}$, we define a function $\omega(a,R^{\rightarrow})$ that returns A if $R^{\rightarrow}$ includes a pair $[a,A]$; it returns a, otherwise. In the following, with a slight abuse of notation, $\omega(A,R^{\rightarrow})$ will denote the application of function ω to each attribute in A.

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

The profile of the relation resulting from a query depends on the profile of the operand relations and on the operators involved in its computation. Every operator only operates on visible attributes

(i.e., attributes in R^{vp} and R^{ve}, which belong to the schema of the operand relation R), but it may affect also implicit, equivalent and derived attributes in the profile of the resulting relation. The approach in [DFJ+17] defined the profiles resulting from the application of projection, selection, cartesian product, join, and group by (this latter with the simplifying assumption of maintaining the same names to attributes over which the aggregate function operates), assuming a simplified profile that does not include component $R^{\rightarrow}$. Indeed, selection, projection, cartesian product, and join do not affect the $R^{\rightarrow}$ component of profiles, which remains unchanged from the operands to the operation result.

The presence of component $R^{\rightarrow}$ in relation profiles permits to easily support the use of rename and set operators, as well as of arithmetic expressions, as illustrated in the following.

Renaming (ρ). Renaming changes the name of a subset of the (plaintext or encrypted) visible attributes of a relation. Since the resulting relation has the same schema as the operand apart from the renamed attributes, the visible component of the profile of the result reflects such a change. Hence, the visible component of the profile of the result is the same as the one of the operand, apart from the fact that the attribute a on which the rename operator is applied is substituted with its new name a' in the plaintext or encrypted component. The implicit attributes and equivalence sets are the same as the ones of the operand. Note that if attribute a appears among the implicit attributes or in an equivalence set in the operand relation, it is not replaced with its new name a'. For each renamed attribute, pair $[a', \omega(a, R^{\rightarrow})]$ is added to the set of renamed attributes. Note that component $R^{\rightarrow}$ keeps track of the correspondence between the new name a' of the attribute and the name of the corresponding attribute in base relations, obtained evaluating $\omega(a, R^{\rightarrow})$. Indeed, if a' is the new name for a and a is not a name in a base relation, a is substituted with $\omega(a, R^{\rightarrow})$ before populating $R^{\rightarrow}$ component in the relation profile. Consider a rename operation $R = \rho_{a' \leftarrow a} R_l$ operating on relation R_l with profile $[R_l^{vp}, R_l^{ve}, R_l^{ip}, R_l^{ie}, R_l^{\simeq}, R_l^{\rightarrow}]$. The profile of R is defined as follows (see Figure 1.3):

- $R^{vp} = R_l^{vp} \cup \{a'\} \setminus \{a\}$ if $a \in R_l^{vp}$, $R^{vp} = R_l^{vp}$ otherwise;
- $R^{ve} = R_l^{ve} \cup \{a'\} \setminus \{a\}$ if $a \in R_l^{ve}$, $R^{ve} = R_l^{ve}$ otherwise;
- $R^{ip} = R_l^{ip}$;
- $R^{ie} = R_l^{ie}$;
- $R^{\simeq} = R_l^{\simeq}$;
- $R^{\rightarrow} = R_l^{\rightarrow} \cup [a', \omega(a, R_l^{\rightarrow})]$.

Figure 1.3 illustrates an example of the profiles resulting from two rename operations, renaming plaintext attribute B (left-hand side of the figure) and encrypted attribute T (right-hand side of the figure) to the new attribute name K.

Set operators ($\cup$, $\cap$, $\setminus$). Set operators are binary operators that work like the corresponding operators from mathematical set theory. They combine the relations resulting from two (or more) queries into a single result set, returning the tuples that belong to: at least one between R_l and R_r (union $\cup$), both R_l and R_r (intersection $\cap$), R_l but not R_r (difference $\setminus$). According to SQL standards, we assume that the schema of the resulting relation is the one of the first operand, that is, R_l. Note that R_l and R_r must have the same number of attributes in their schema to enable

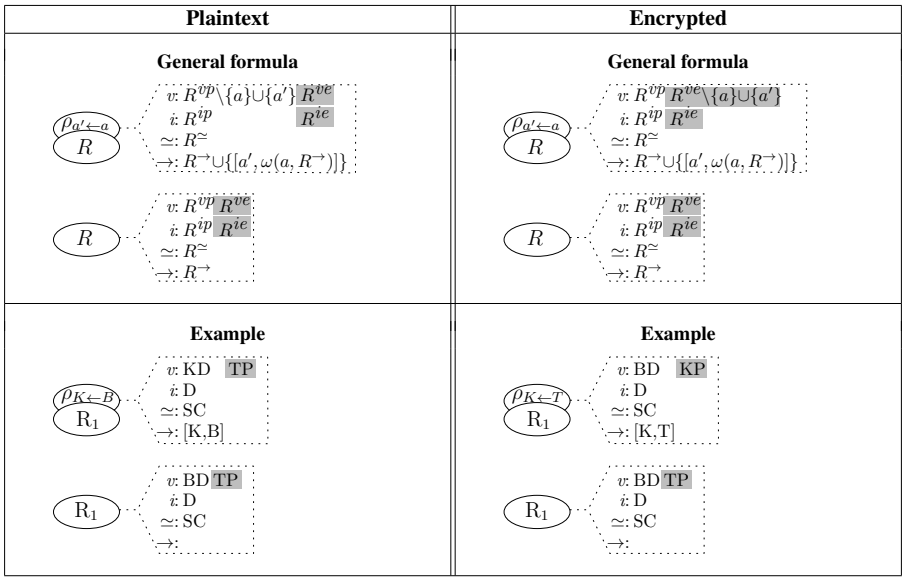


Figure 1.3: Graphical representation of the profile resulting from rename operation

the evaluation of a set operator. Since set operators produce as a result a relation with the same schema as the first operator R_l, the result has the same visible attributes as R_l. However, since the result conveys information about both the operands, the implicit attributes, the sets of equivalent attributes, and renamed attributes are the union of the corresponding components (i.e., sets of attributes) in the profiles of the operands. Also, for each pair of attributes a_{l_i} in R_l and a_{r_i} in R_r appearing in the same (i-th) position in the schema of the operand relation, equivalence $\{a_{l_i}, a_{r_i}\}$ is added to the equivalence set. Indeed, the i-th attribute in the schema of the result is obtained by comparing and/or combining the values of the i-th attribute a_{l_i} in the schema of R_l and the values of the i-th attribute a_{r_i} in the schema of R_r. More precisely, union operator appends the values in a_{r_i} to the ones in a_{l_i}, while intersection and difference operators compare the values in a_{l_i} with the ones in a_{r_i}. It is interesting to note that even if intersection and difference operators reduce the number of tuples in the resulting relation with respect to the number of tuples in its operands, their implicit information (and hence their profiles) are richer than those of its operands singularly taken. For instance, consider a relation R resulting from the difference $R = R_l \setminus R_r$. Clearly, R strongly depends on R_r even if no tuple in R_r appears in R (and hence its implicit components need to consider those of both R_l and R_r).

Consider operation $R = R_l \; set\_op \; R_r$, with $set\_op \in \{\cup, \cap, \setminus\}$ and operating on relations R_l and R_r with profile $[R_l^{vp}, R_l^{ve}, R_l^{ip}, R_l^{ie}, R_l^{\simeq}, R_l^{\rightarrow}]$ and $[R_r^{vp}, R_r^{ve}, R_r^{ip}, R_r^{ie}, R_r^{\simeq}, R_r^{\rightarrow}]$, respectively. The profile of R is defined as follows (see Figure 1.4):

- $R^{vp} = R_l^{vp}$;
- $R^{ve} = R_l^{ve}$;
- $R^{ip} = R_l^{ip} \cup R_r^{ip}$;
- $R^{ie} = R_l^{ie} \cup R_r^{ie}$;
- $R^{\simeq} = R_l^{\simeq} \cup R_r^{\simeq} \cup \{\{a_{l_i}, a_{r_i}\} : a_{l_i} \in R_l^{vp} \cup R_l^{ve}, a_{r_i} \in R_r^{vp} \cup R_r^{ve}, i = 1, \ldots, |R_l^{vp} \cup R_l^{ve}|\}$;
- $R^{\rightarrow} = R_l^{\rightarrow} \cup R_r^{\rightarrow}$.

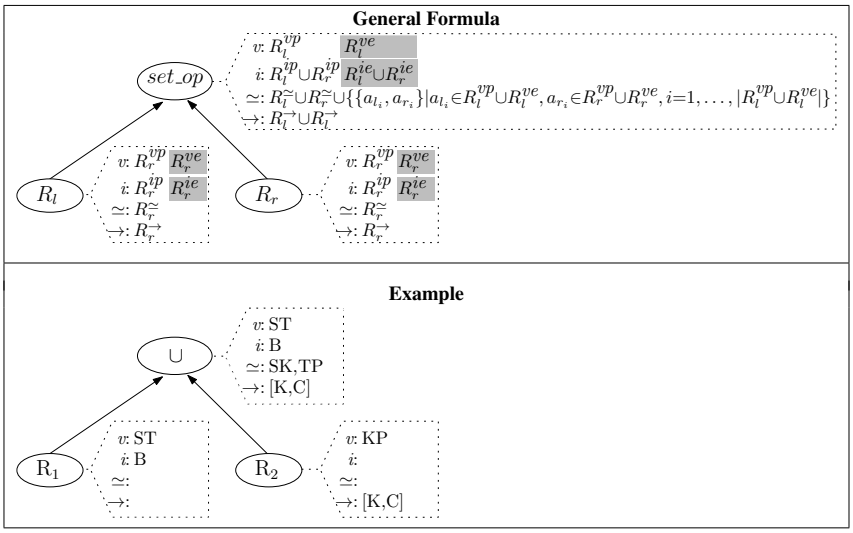


Figure 1.4: Graphical representation of the profile resulting from set operators

Note that the representation of attributes a_{l_i} and a_{r_i} must be consistent, that is, the attributes must be both plaintext or both encrypted for the evaluation of set operators. Figure 1.4 illustrates the profile resulting from the execution of the union operator over two relations with schemas composed of two attributes. The resulting profile, while keeping in its schema the attributes of the first operand, includes in its equivalent component the fact that SK and TP have been connected in the computation (where K is a renamed version of attribute C).

Arithmetic expressions and aggregate functions. The result of a query can include, besides a list of attributes, also the result of an arbitrary arithmetic expression or of an aggregate function over a set of attributes. The query associates a name a with the result of the arithmetic expression Exp defined over a set A of attributes, or of the aggregate function Agg operating over a set A of attributes. We denote such operations with notations $\rho_{a \leftarrow Exp}(R)$ and $\gamma_{X,a \leftarrow f(Y)}(R)$, respectively. Note that the attributes in A must be all encrypted or all plaintext for the evaluation of the arithmetic expression or of the aggregation function. Clearly, iff an operation requires to operate on plaintext values, attributes in A must be plaintext.

Consider the evaluation of arithmetic expression Exp defined over a set A of attributes to which the query associates name a, denoted $\rho_{a \leftarrow Exp}(R_l)$. The evaluation of $R = \rho_{a \leftarrow Exp}(R_l)$, operating on relation R_l with profile $[R_l^{vp}, R_l^{ve}, R_l^{ip}, R_l^{ie}, R_l^{\simeq}, R_l^{\rightarrow}]$, produces a relation R with profile:

- $R^{vp} = R_l^{vp} \setminus A \cup \{a\}$ if $A \subseteq R_l^{vp}$, $R^{vp} = R_l^{vp}$ otherwise;
- $R^{ve} = R_l^{ve} \setminus A \cup \{a\}$ if $A \subseteq R_l^{ve}$, $R^{ve} = R_l^{ve}$ otherwise;
- $R^{ip} = R_l^{ip}$;
- $R^{ie} = R_l^{ie}$;
- $R^{\simeq} = R_l^{\simeq}$;
- $R^{\rightarrow} = R_l^{\rightarrow} \cup [a, \omega(A, R_l^{\rightarrow})]$

Figure 1.5 illustrates an example of the profiles resulting from two arithmetic expressions, renaming to a new attribute with name K the sum computed over two plaintext (left-hand side of the figure) and two encrypted (right-hand side of the figure) attributes T and P.

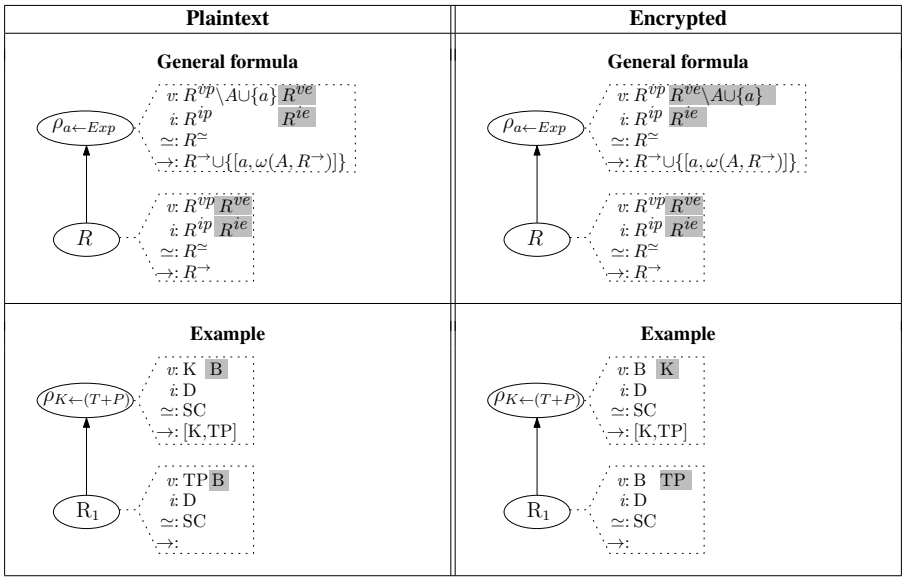


Figure 1.5: Graphical representation of the profile resulting from arithmetic expressions

Consider now the evaluation of an aggregate function f defined over a set Y of attributes and computed over groups of tuples with the same values for attributes in X, denoted $\gamma_{X,a\leftarrow f(Y)}(R_l)$. The evaluation of $R = \gamma_{X,a\leftarrow f(Y)}(R_l)$, operating on relation R_l with profile $[R_l^{vp}, R_l^{ve}, R_l^{ip}, R_l^{ie}, R_l^{\simeq}, R_l^{\rightarrow}]$, produces a relation R with profile:

- $R^{vp} = (R_l^{vp} \cap X) \cup \{a\}$ if $Y \subseteq R_l^{vp}$, $R^{vp} = (R_l^{vp} \cap X)$ otherwise;
- $R^{ve} = (R_l^{ve} \cap X) \cup \{a\}$ if $Y \subseteq R_l^{ve}$, $R^{ve} = (R_l^{ve} \cap X)$ otherwise;
- $R^{ip} = R_l^{ip} \cup (R_l^{vp} \cap X)$;
- $R^{ie} = R_l^{ie} \cup (R_l^{ve} \cap X)$;
- $R^{\simeq} = R_l^{\simeq}$;
- $R^{\rightarrow} = R_l^{\rightarrow} \cup [a, \omega(Y, R_l^{\rightarrow})]$.

Figure 1.6 illustrates an example of the profiles resulting from two aggregate functions, renaming to a new attribute with name K the average computed over the plaintext (left-hand side of the figure) and encrypted (right-hand side of the figure) attribute P.

1.5 Authorization enforcement

In this section, we illustrate how an authorization policy can be enforced (Section 1.5.1) and how the execution of the operations entailed by a query can be assigned to subjects in the respect of such policy (Section 1.5.2).

1.5.1 Authorized visibility

Given a relation R with profile $[R^{vp}, R^{ve}, R^{ip}, R^{ie}, R^{\simeq}, R^{\rightarrow}]$, the release of R to a subject S depends on the set of authorizations granted to S. However, authorizations are defined only on base attributes, that is, those appearing in the base relations. Consistently with the assumption of operating under

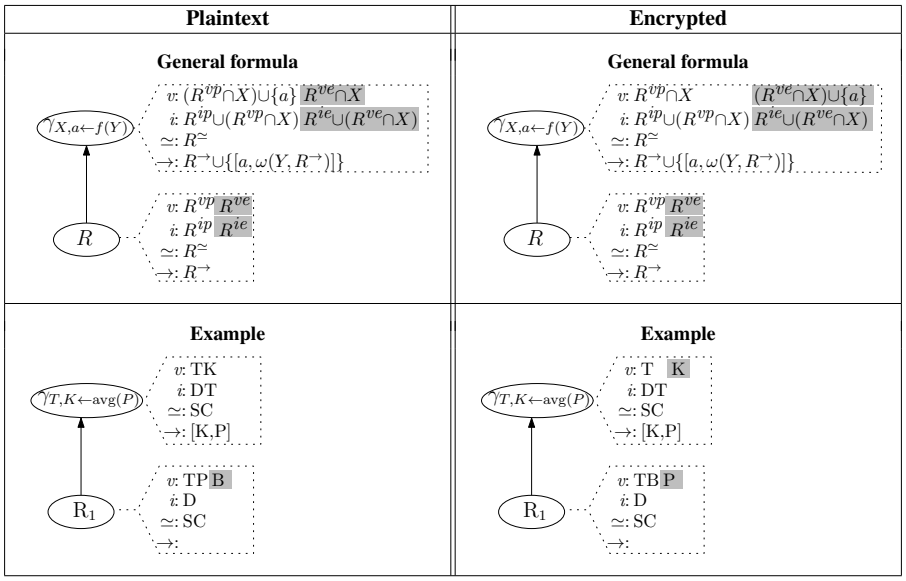


Figure 1.6: Graphical representation of the profile resulting from aggregate functions

a closed policy, the release of any derived attribute in R would be denied. This situation is clearly unacceptable, since the values of the derived attributes depend on the values of the base attributes from which they have been (directly or indirectly) computed. Hence, we need to define a way for regulating their release while operating with authorizations including only base attributes. Our approach relies in the translation of relation profiles into *simplified relation profiles*, that is, relation profiles defined over base attributes only. To this end, we leverage function $\omega(a,R^{\rightarrow})$ (Section 1.4) to replace derived attributes with the corresponding base attributes on which they have been defined. Hence, a simplified relation profile is composed of five (rather than six) components, since component $R^{\rightarrow}$ is omitted and its content used to transform the other components in such a way to include base attributes only. Given a relation profile $[R^{vp},R^{ve},R^{ip},R^{ie},R^{\simeq},R^{\rightarrow}]$, its simplified profile is obtained by:

1. substituting each occurrence of attribute a in R^{vp}, R^{ve}, R^{ip}, R^{ie}, and $R^{\simeq}$ with $\omega(a,R^{\rightarrow})$;
2. inserting A into $R^{\simeq}$ for each pair $[a,A]$ in $R^{\rightarrow}$.

The first step simply substitutes each derived attribute in the relation profile with the set of attributes from which it has been derived. The second step instead includes additional equivalences among attributes in the $R^{\simeq}$ component. This is needed to impose that a subject can access an attribute a derived from a set A of other attributes only when it has the same visibility on all attributes in A (more details on this uniform visibility requirement will be illustrated in the remainder of this section). For instance, assume that attribute a_{ij} is defined as the sum of attributes a_i and a_j. The visibility of a subject S over a_{ij} should be the minimum between her visibility over a_i and her visibility over a_j (i.e., if S can access a_i plaintext and a_j encrypted, she cannot access their sum in plaintext as she could reconstruct a_j). However, we also require the same visibility over all attributes involved in a computation because, for example, even the encrypted visibility over the result of a computation, combined with the plaintext visibility of one of its operands, may cause improper information leakage.

Formally, the simplified profile of a relation is defined as follows.

Definition 1.5.1 (Simplified Relation Profile) *Let R be a relation with profile $[R^{vp}, R^{ve}, R^{ip}, R^{ie}, R^{\simeq}, R^{\rightarrow}]$. The* simplified profile *of R is a 5-tuple of the form $[R^{vp}_{\rightarrow}, R^{ve}_{\rightarrow}, R^{ip}_{\rightarrow}, R^{ie}_{\rightarrow}, R^{\simeq}_{\rightarrow}]$ where $R^{vp}_{\rightarrow} = \omega(R^{vp}, R^{\rightarrow})$, $R^{ve}_{\rightarrow} = \omega(R^{ve}, R^{\rightarrow})$, $R^{ip}_{\rightarrow} = \omega(R^{ip}, R^{\rightarrow})$, $R^{ie}_{\rightarrow} = \omega(R^{ie}, R^{\rightarrow})$, and $R^{\simeq}_{\rightarrow} = \omega(R^{\simeq}, R^{\rightarrow}) \cup \{A, \forall [a,A] \in R^{\rightarrow}\}$.*

The simplified profile of a relation is *equivalent* to its relation profile (meaning that it conveys the same informative content), since it replaces the names of derived attributes with the names of the base attributes on which they depend. Note that, given a query plan, the simplification of the profile of the relation R_x resulting from the evaluation of a node n_x can be performed only after the evaluation of the operator represented by the parent of node n_x (i.e., the profile of the parent of n_x should operate on the original, non simplified, profile), because such an operator might operate over the derived attributes. For instance, consider the relation profiles resulting from the aggregate functions in Figure 1.6: the aggregate *avg(P)* (regardless of P being plaintext or encrypted), is renamed to K, and hence possible subsequent operations are expected to be defined over the newly introduced K.

Having defined simplified relation profiles, including only attributes over which authorizations have been defined, we can verify the satisfaction of the authorization policy by directly comparing simplified profiles and authorizations. For the sake of readability, in the following we denote with notation $\mathscr{P}_S$ ($\mathscr{E}_S$, resp.) the set of attributes that S can access in plaintext (in encrypted form, resp.). To verify the satisfaction of the authorization policy in the release of a relation, we define the concept of *authorized relation* as follows.

Definition 1.5.2 (Authorized Relation) *Let R be a relation with simplified profile $[R^{vp}_{\rightarrow}, R^{ve}_{\rightarrow}, R^{ip}_{\rightarrow}, R^{ie}_{\rightarrow}, R^{\simeq}_{\rightarrow}]$. A subject S is* authorized *for R iff:*

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

According to Definition 1.5.2, a subject S is authorized to access a relation R iff: *1) S* is authorized to access in plaintext all the (visible or implicit) attributes represented in plaintext in R; *2) S* is authorized to access in plaintext or in encrypted form all the (visible or implicit) attributes represented in encrypted form in R; *3) S* is authorized to access in the same form (either plaintext or encrypted) all the equivalent attributes, that is, attributes that appear together in an equivalence set in $R^{\simeq}$ (uniform visibility). Note that the last condition enforces a control on the indirect information flows caused by equivalence relationships produced in the evaluation of the query computation, to prevent unauthorized exposure of information. Clearly, the condition requires that a subject S be authorized to access all equivalent attributes. The condition also imposes that S enjoys the same (plaintext or encrypted) visibility over all attributes in an equivalence set, for all sets. This prevents the information leakage that may flow to a subject with different visibilities allowed over the attributes in a set: to illustrate, consider a relation with $[a_1, a_2, \_, \_, \{a_1, a_2\}]$, and a subject S allowed to access a_1 plaintext and a_2 encrypted. If S were to be given access to the relation, she might leverage her plaintext visibility over a_1 to determine the (plaintext) values of a_2, violating the authorizations. Requiring uniform visibility prevents such inference channel.

Definition 1.5.2 illustrates the conditions that must be satisfied by a subject to access a relation. Our problem however concerns assigning subjects the *execution of an operation* in a query tree

plan. Each operation takes as input a relation (or two relations, in case of join, cartesian product, and set operators), and produces a relation. A subject is authorized to perform an operation if it is authorized to access both its operands and its result. Indeed, the profile of the relations input and output of an operation capture all the information flows caused by the evaluation of the operation.

1.5.2 Assigning operations to subjects

While original query plans assume all the attributes to be presented in plaintext and do not include encryption/decryption operations, our authorization model distinguishes between two possible views over the data: plaintext and encrypted. We then leverage *on-the-fly encryption* to adjust the visibility over the attributes to allow subjects authorized for encrypted visibility only to participate in query evaluation. Indeed, on-the-fly encryption can be used to hide plaintext values of an attribute when the subject involved for a computation over it can access it in encrypted form only. Similarly, on-the-fly decryption can be adopted whenever dictated by operation execution. Indeed, some operations cannot be evaluated over encrypted data. Also, set operators specifically require that corresponding attributes in the operand relations be represented in the same form to enable the evaluation of the operation. The evaluation of intersection and difference requires to compare the tuples in the two operands to compute the result. Even if the evaluation of union operator does not require such a comparison, it would still not be possible to operate on the resulting relation where different tuples have a different representation form for a same attribute.

We enrich query plans with encryption and decryption operations, so to adjust attribute visibility as demanded by authorizations enforcement and operations execution. We refer to a query plan enriched with on-the-fly encryption and decryption operation as an *extended query plan*. To illustrate, consider the two relations Hospital and Insurance illustrated in Section 1.2, and the following query: "SELECT T, avg(P) FROM HOSP JOIN INS ON S=C WHERE D='stroke' GROUP BY T HAVING avg(P)>100" retrieving, for each treatment given to patients hospitalized for stroke, the average insurance premium (if greater than USD100). Figure 1.1(a) illustrates an example of an extended plan for this query. Encryption and decryption operations are denoted with grey and white boxes, respectively, surrounding the encrypted/decrypted attributes. The plan in the figure includes two encryption operations (over S and over CP before the computation of the join) and one decryption operation (over P before the computation of the last selection $\sigma_{\text{avg}(P)<100}$). Nodes in the figure are also enriched with the simplified profiles resulting from their operations, and with the subjects in charge for their execution (reported on the left-hand side of each node).

Given a query plan, there can exist several extended query plans and, for each extended query plan, different assignments of subjects to operations that satisfy authorizations. An extreme strategy consists in keeping all attributes plaintext, but such a solution limits the number of subjects to which each operation can be assigned. On the other hand, keeping all the attributes encrypted would prevent the evaluation of operations requiring plaintext visibility over attributes. To determine an extended query plan that enables the identification of subjects that can perform operations without violation authorizations, we associate each node in the query plan with its *minimum required views*. The minimum required view associated with a node represents the profile of the node obtained assuming that all the attributes are encrypted with the exception of those needed plaintext for the evaluation of the operation. Intuitively, minimum required views characterize operation requirements that can limit or prevent the adoption of encryption. All subjects that are authorized for the minimum required view associated with a node can be considered *candidates* for executing operation (i.e., they can be assigned the evaluation of the operation in the respect of au-

thorizations). Operations can be assigned to *any* candidate, since visibility over the attributes can always be adjusted by inserting on-the-fly encryption operations as demanded by authorizations, without affecting the evaluation of the operation. For instance, with reference to the extended plan in Figure 1.1, the execution of the join operation can be assigned to subject $\mathbb{X}$ thanks to the fact that attributes S, C, and P (over which $\mathbb{X}$ is only authorized for the encrypted visibility) are encrypted before the join execution.

Several strategies can be adopted for choosing the most suitable candidate for each operation in a query plan. If the cost of encryption can be considered negligible, any query optimizer can be used for selecting the best candidate. Encryption and decryption are then inserted when needed to regulate visibility and to allow operation execution. When instead the impact of encryption and decryption operations is expected to be non-negligible (for instance, when adopting advanced encryption schemes that are computationally intensive and which could significantly impact the size of the input data), those costs should be taken into consideration to the aim of finding an assignment of operations to candidates that minimizes the overall execution costs. This latter strategy is currently under investigation, to the aim of defining appropriate cost functions able to consider encryption and decryption costs in the computation of the most suitable candidate for each operation in the considered query plan.

1.6 Summary

This chapter focused on the problem of collaborative computations for analyzing data from different sources in the data market, assuming different trust by the data owners in the different subjects involved in the computation. The approach illustrated in this chapter enables the data market to rely on different providers for delegating the analysis and combination of data stored in the market, possibly owned by different authorities. The proposed solution leverages on the possibility of expressing authorizations regulating the (plaintext or encrypted) visibility to enable collaborative query evaluation.

MOSAICrOWN is now studying a novel solution for minimizing the economic cost of collaborative query evaluation, taking into consideration the overhead caused by encryption and decryption operations.

2. AONT with splitting and fragmentation

In this chapter, we discuss the problems of storing and managing resources in the data market, providing availability and security guarantees in scenarios where the data market relies on Decentralized Cloud Storage (DCS) platforms. DCS services represent a promising opportunity for the data market for storing resources. DCS services rely on the availability of multiple nodes that can be used to store resources in a decentralized manner. In such services, individual resources are fragmented in slices allocated (with replication to provide availability guarantees) to different nodes. The main characteristics of a DCS is the cooperative and dynamic structure formed by independent nodes (providing a multi-authority storage network) that can join the service and offer storage space, typically in exchange of some reward. The use of a dynamic network of unknown (and potentially non-trusted) peers clearly introduces the problem of guaranteeing proper protection to resources, ensuring their availability and security (for both confidentiality and integrity). In fact, a DCS is a potentially unstable network, and hence continuous participation of every single node cannot be assumed. Given this, and in line with the distributed nature of DCS services, resources are sliced into many slices, with different slices allocated to different nodes of the network, with replication to guarantee availability. Reconstruction of a resource requires collecting the different slices composing it. Also, nodes participating in the DCS - which can dynamically join and leave and are anonymous - cannot be considered trusted, hence resource confidentiality needs to be protected against each of them (as well as against possible coalitions of malicious nodes) and the data market should be able to assess integrity of the slices (and hence resources).

In this chapter, we present a solution to enable data markets to securely store the resources it manages in DCS services. While client-side encryption provides a first crucial layer of protection in DCS, it leaves resources exposed to threats, especially in the long term. For instance, resources are still vulnerable in case the encryption key is exposed (e.g., as a consequence of access revocation), or in case of malicious nodes not deleting their slices upon request for resource deletion. For this reason, we leverage the protection guarantees offered by *All-Or-Nothing-Transform* (AONT). We devise an approach to carefully control *resource slicing* and *allocation* to nodes in the DCS network, with the goal of ensuring both *availability* and *security*. We rely on replication and on erasure codes to guarantee availability, that is, the ability for authorized users to retrieve all the slices to reconstruct the resource. We rely on AONT to guarantee security, that is, to protect against malicious parties jointly collecting their slices to prevent resource deletion or attempt decryption in case of key exposure. In this chapter, we use the term *slicing* to refer to the cutting of a resource and the term *slices* to refer to the result of such a process. A slice is therefore a chunk of the resource and represents a unit of allocation.

The remainder of this chapter is organized as follows. Section 2.1 presents related works and the innovation of MOSAICrOWN. Section 2.2 introduces the basic concepts. Section 2.3 defines the properties of a decentralized allocation function with respect to replication and protection. Section 2.4 discusses slicing and allocation strategies. Section 2.5 illustrates availability and security guarantees and discusses the setting of parameters guiding slicing and allocation. Section 2.6

presents the combined adoption of AONT and error correcting code techniques to provide for better availability guarantees. Section 2.7 analyzes the availability and security guarantees obtained when combining AONT and fountain codes. Section 2.8 concludes the chapter.

2.1 State of the art and MOSAICrOWN innovation

In this section, we illustrate the state of the art and the innovation produced by MOSAICrOWN for the storage of resources managed in the data market, relying on DCS services.

2.1.1 State of the art

RAID [PGK88] is one of the main contributions aimed at the construction of reliable systems. RAID is normally deployed on local drives. With the advent of the cloud, RAID has been extended to take adversarial failures into consideration. Along this line of works, *HAIL (High-Availability and Integrity Layer)* [BJO09a] extended RAID with multiple cloud storage providers and a *Proof of Retrievability (POR)* [BJO09b] scheme to verify that a provider still holds a certain piece of information. *HAIL* is however not well-suited for DCS systems. Also, *HAIL* does not take into account the possibility of adversarial users trying to reconstruct resources for their personal profit.

Many DCS networks that have recently been proposed already include a certain degree of security guarantees (i.e., protection against malicious parties jointly collecting all the slices composing a resource). Among them, Storj [WBB+16] and Sia [VC14] adopt client-side encryption and do not protect the outsourced data against coalitions of malicious nodes. SAFE Network [Irv10] instead adopts a self-encryption technique: the resource is divided into shards and a weak AONT among three shards is applied before uploading them. In [PHI14] the design of the SAFE Network and the possible attack vectors are analyzed. The solution proposed in [Irv10, PHI14] is predetermined and the interaction between redundancy and security is not analyzed.

Another related line of works is security of outsourced data (e.g., [AJJP16, ATL15, NAL14, TPPG13]), which can be improved using AONT. Existing solutions however consider domains different from DCS.

A precursor of DCS is represented by P2P systems. The P2P system closer to our proposal, which considers reliability and security, is Tangler [WM01]. The goal of Tangler is censorship resistance, which is a potential application of DCS, but not its main goal. A characteristic of Tangler is the use of Shamir's method, which is quite expensive in terms of storage and bandwidth. Also, it does not aim at combining availability and confidentiality requirements in data allocation.

The combined adoption of AONT and error-correcting code techniques has been recently explored to the aim of protecting outsourced data against possibly curious storage providers and offering high performance (e.g., [BMMC14, RP11]). The proposal *AONT-RS* [RP11] combines Rivest's AONT [Ron97] with Reed-Solomon, while *AONT-LT* [BMMC14] uses Luby Transform code (which is a class of fountain codes [CSGM06, Sho11]) instead of Reed-Solomon. These proposals are specifically focused on static scenarios, where the set of nodes is fixed and nodes are not expected to frequently leave/join the network. These solutions are then not suited to DCS scenarios.

2.1.2 MOSAICrOWN innovation

MOSAICrOWN produced several advancements over the state of the art, which are discussed in this section.

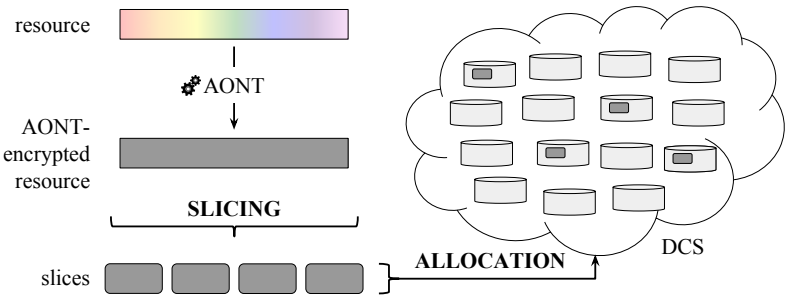


Figure 2.1: Reference scenario

- The first innovation is represented by a technique enabling the data market to control resource slicing and allocation to nodes in a DCS, used for data storage, in such a way to guarantee both availability and security. The proposed solution leverages the guarantees offered by AONT, combined with resource slicing and slice replication.
- The second innovation is represented by the definition of different strategies for slicing and distributing resources across the decentralized network, and the analysis of their characteristics in terms of availability and security guarantees. The modeling of the slicing ad allocation problem enables the data market to control the granularity of slicing and the diversification of allocation to ensure the aimed availability and security guarantees.
- The third innovation is represented by the combined adoption of AONT and fountain codes for better security and availability guarantees, while reducing the performance overhead of current solutions, in scenarios where the nodes composing the DCS are not stable and can dynamically join and leave the network.

The results obtained by MOSAICrOWN and illustrated in this chapter have been published in [BDF+19b, BDF+20]. A preliminary version of the AONT-based approach illustrated in this chapter has been implemented by one of the tools presented in deliverable D4.1 "First version of encryption-based protection tools" [FL20].

2.2 Preliminaries

The two basic building blocks enabling the development of our solution are the adoption, at the client side, of *All-or-Nothing-Transform* (AONT) and of *fountain codes*. Figure 2.1 illustrates our reference scenario. The focus of this chapter is the design of proper *slicing* of resources and the *allocation* of the produced slices to different nodes in the DCS system. The chapter will also illustrate the use of fountain codes, in combination with AONT, to provide availability guarantees in case of node failure.

2.2.1 AONT

AONT is an encryption mode that requires the use of an encryption key. The encryption driven by the key represents the primary protection, and the use of AONT encryption mode further strengthens security. An AONT-encryption mode transforms a plaintext resource into a ciphertext, with the property that the whole result of the transformation is required to obtain back the original

plaintext. Indeed, absence of even a single block from the ciphertext prevents decryption of any other block. AONT guarantees in fact complete interdependence (*mixing*) among the bits of the encrypted resource in such a way that the unavailability of a portion of the encrypted resource prevents the reconstruction of any portion of the original plaintext. A party having access to a portion of the encrypted resource (but not to the encrypted resource in its entirety):

- if knowing the encryption key, it will not be able to reconstruct any portion of the resource (i.e., it will not be able to derive any information from the AONT-encrypted portions it has; the only option would be to attempt a brute force attack on the possible configurations of the missing portions, but their possibly large size makes this attack unfeasible);
- if not knowing the encryption key, it will not be able to perform brute-force attacks for guessing such a key, as any key (even the correct one) will be ineffective if not applied to the complete resource. AONT protection schemes can be built with the use of common cryptographic functions, like symmetric encryption and hash functions.

An example of an AONT scheme that guarantees complete mixing, which is at the basis of the solutions presented in this chapter, is Mix&Slice [BDF+16].

The basic building block of Mix&Slice is the application of a symmetric block cipher operating on blocks, which guarantees complete dependency of the encrypted result from every bit of the input and the impossibility, when missing some bits of an encrypted version of a block, to retrieve the original plaintext block (even if parts of it are known). The larger the number of bits that are missing, the harder the effort required to perform a brute-force attack, which requires attempting 2^x possible combinations of values when x bits are missing. Such *security parameter* is at the center of our approach and we explicitly identify a sequence of bits of its length as the atomic unit on which our approach operates, which we call *mini-block*. Applying block encryption with explicit consideration of such atomic unit of protection, and extending it to a coarser-grain with iterative rounds, our approach identifies the following basic concepts.

- *Block*: a sequence of bits input to a block cipher.
- *Mini-block*: a sequence of bits, of a specified length (divisor of the size of the block), contained in a block. It represents our *atomic unit* of protection.
- *Macro-block*: a sequence of blocks. It allows extending the application of block cipher on sequences of bits larger than individual blocks.

The basic step of Mix&Slice (on which it iteratively builds to provide complete mixing within a macro-block) is the application of encryption at the block level. This application is visible at the top of Figure 2.2, where the first row reports a sequence of 16 mini-blocks ($[0], \ldots, [15]$) composing 4 blocks. The second row is the result of block encryption on the sequence of mini-blocks. As illustrated by the pattern-coding in the figure, encryption provides mixing within each block so that each mini-block in the result is dependent on every mini-block in the same input block. One round of block encryption provides mixing only at the level of block.

The idea of Mix&Slice is to extend mixing to the whole macro-block by the iterative application of block encryption on, at each round, blocks composed of mini-blocks that are representative of different encryptions in the previous round (i.e., mini-blocks that belong to the block resulting from the encryption of a block in the previous encryption round). For instance, with reference to Figure 2.2, where $[0]_1, \ldots, [15]_1$ are the mini-blocks resulting from the first round, the second

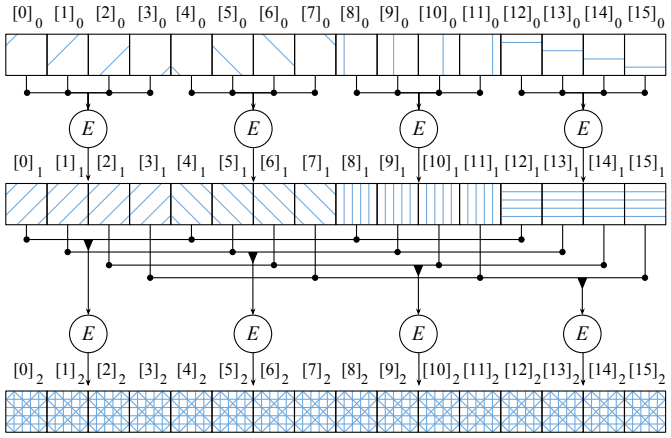


Figure 2.2: An example of mixing of 16 mini-blocks assuming $m = 4$

round would apply again block encryption, considering different blocks each composed of a representative of a different computation in the first round (i.e., one mini-block among the first four, one among the second four, one among the third four, and one among the last four). To guarantee such a composition, Mix&Slice defines the blocks input to the four encryption operations as composed of mini-blocks that are at distance 4 in the sequence, which corresponds to say that they resulted from different encryption operations in the previous round. The blocks considered for encryption would then be $\langle [0]_1[4]_1[8]_1[12]_1\rangle$, $\langle [1]_1[5]_1[9]_1[13]_1\rangle$,$\langle [2]_1[6]_1[10]_1[14]_1\rangle$,$\langle [3]_1[7]_1[11]_1[15]_1\rangle$. The result would be a sequence of 16 mini-blocks, each of which is dependent on each of the 16 original mini-blocks, that is, the result provides mixing among all 16 mini-blocks, as visible from the pattern-coding in the figure. With 16 mini-blocks, two rounds of encryption suffice for guaranteeing mixing among all of them. Providing mixing for larger sequences clearly requires more rounds. More precisely, at each round i, mini-blocks are mixed among chunks of m^i mini-blocks (with m the number of mini-blocks in a block, 4 in our example), hence ensuring at round i, mixing of a macro-block composed of m^i mini-blocks.

An important feature of the mixing is that the number of bits that are passed from each block in a round to each block in the next round is equal to the size of the mini-block. This guarantees that the uncertainty introduced by the absence of a mini-block at the first round maps to the same level of uncertainty for each of the blocks involved in the second round, and iteratively to the next rounds, thanks to the use of AES at each iteration. This implies that a complete mixing of the macro-block requires at least $\log_m(m \cdot b)$ rounds, that is, the rounds requested by our technique.

Another crucial aspect is that the representation of the resource after each round (i.e., the output of each round) has to be of the same size as the original macro-block. In fact, if the transformation produced a more compact representation, there would be a possibility for a user to store this compact representation and maintain access to the resource even after revocation to weaken the security guarantees provided by the encryption mode.

When resources are extremely large (or when access to a resource involves only a portion of it) considering a whole resource as a single macro-block may be not desirable. Even if only with a logarithmic dependence, the larger the macro-block the more the encryption (and therefore decryption to retrieve the plaintext) rounds required. Also, encrypting the whole resource as a single macro-block implies its complete download at every access, when this might actually not be needed for service. Accounting for this, we do not assume a resource to correspond to an individual macro-block, but assume instead that any resource can be partitioned into M macro-

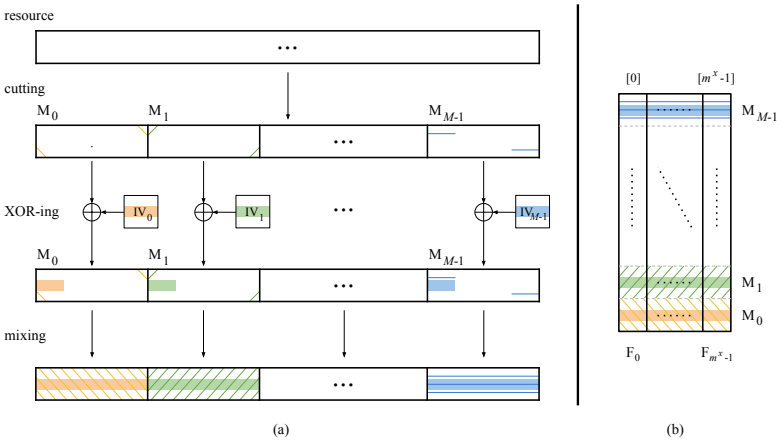


Figure 2.3: From resource to fragments

blocks, which can then be mixed independently. Encryption of a resource would then entail a preliminary step cutting the resource in different, equally sized, macro-blocks on which mixing operates. To ensure the mixed versions of macro-blocks be all different, even if with the same original content, the first block of every macro-block is XORed with an *initialization vector* (*IV*).

The starting point for introducing mixing is to ensure that each single bit in the encrypted version of a macro-block depends on every other bit of its plaintext representation, and therefore that removing any one of the bits of the encrypted macro-block would make it impossible (apart from brute-force attacks) to reconstruct any portion of the plaintext macro-block. Such a property operates at the level of macro-block. Hence, if a resource (because of size or need of efficient fine-grained access) has been partitioned into different macro-blocks, removal of a mini-block would only guarantee protection of the macro-block to which it belongs, while not preventing reconstruction of the other macro-blocks (and therefore partial reconstructions of the resource). Resource protection can be achieved if, for each macro-block of which the resource is composed, a mini-block is removed. Slicing the encrypted resource consists in defining different *fragments* such that a fragment contains a mini-block for each macro-block of the resource, no two fragments contain the same mini-block, and for every mini-block there is a fragment that contains it. To ensure all this, as well as to simplify management, Mix&Slice slices the resource simply putting in the same fragment the mini-blocks that occur at the same position in the different macro-blocks. Figure 2.3(b) illustrates the slicing process

2.2.2 Fountain codes

Fountain codes are a class of erasure codes preventing that the loss of one of the transmitted or stored blocks of a resource causes a data loss. Given a resource r, partitioned into f different *fragments*, an erasure code generates a set of $s>f$ *encoded slices* that depend on the resource content and support the reconstruction of r through the combination of a subset of the encoded slices. Fountain codes, unlike other erasure codes (e.g., Reed-Solomon [RS60]), offer probabilistic reconstruction guarantees, meaning that with a probability $p < 1$, s of the f slices are sufficient for reconstructing r. The reconstruction probability p exponentially increases by retrieving additional slices. Although probabilistic, fountain codes have two main characteristics that, as we will discuss in the next section, allow us to profitably use them in the DCS context. First, they are *rateless*, that is, using these codes it is possible to create a new (i.e., different from each other) slice on the fly

and therefore the number s of encoded slices is not fixed a priori. Independently from the number s of slices, any subset of (at least) f slices can be used to reconstruct r. Second, each slice depends on a subset of (and not on all) the f original fragments of the resource and then only a subset of the original fragments are needed for generating a new slice.

2.3 Allocation properties

In our approach, the slicing of the resources into several slices to be distributed at the different nodes of the data market is guided by the availability and protection properties that need to be guaranteed. Availability (despite nodes failure or temporary unreachability) is provided through replication, security is provided through protection against malicious coalitions. Malicious nodes (and coalitions thereof) are interested in making the resource unavailable, by not returning the slices of the resource they store, or in providing access to a resource even after its deletion, by not removing the slices of the resource they store and returning such slices to (not authorized) subjects who pay for it. Before addressing slicing, we then characterize the replication and coalition resistance properties of the distribution of a resource.

We assume a (transformed) resource that has undergone AONT encryption (as described in the previous section) at the client side. For simplicity, we will omit such an explicit remark on transformation, and we will simply use the term resource to denote an AONT-encrypted resource. Also, we assume a resource to be composed of different slices, for storage in a data market relying on a DCS. We will address the problem of producing such slices in Section 2.4.

We model a resource as a set $\mathscr{S} = \{\mathtt{s}_1, \ldots, \mathtt{s}_\mathtt{s}\}$ of slices to be allocated to the nodes, denoted $\mathscr{N}$, of the DCS. The following definition formalizes slice allocation.

Definition 2.3.1 (Allocation function) *Let $\mathscr{S}$ be a set of slices composing a resource and $\mathscr{N}$ be a set of nodes. An* allocation function $\varphi : \mathscr{S} \rightarrow 2^{\mathscr{N}} \setminus \emptyset$ *assigns each slice $\mathtt{s}_\mathtt{i} \in \mathscr{S}$ to a set of nodes $\varphi(\mathtt{s}_\mathtt{i}) = \mathtt{N}_\mathtt{i} \subseteq \mathscr{N}$, $\mathtt{N}_\mathtt{i} \neq \emptyset$.*

The allocation function dictates how slices are allocated to nodes in the DCS. The consideration of sets of nodes (in contrast to individual nodes) in the co-domain accommodates replication. The exclusion of the empty set of nodes ensures lossless distribution (i.e., each slice is allocated to at least one node). Figure 2.4 illustrates an example of an allocation function, considering a resource split into ten slices ($\mathscr{S} = \{\mathtt{s}_1, \ldots, \mathtt{s}_{10}\}$) allocated to five nodes ($\mathtt{n}_1, \ldots, \mathtt{n}_5$) in the DCS (nodes not used in the allocation are not reported in the figure). The figure has a row for each node and a column for each slice. The allocation of a slice to a node is represented by a gray box at the intersection between the row representing the node and the column representing the slice. Empty boxes with a dotted frame represent the fact that the slice is not allocated to the node. For example, $\varphi(\mathtt{s}_1) = \{\mathtt{n}_1, \mathtt{n}_2\}$.

We identify two main properties of an allocation, characterizing the availability, provided by *replication*, and the *protection* against possible malicious coalitions of nodes, provided by the diversification of the allocation.

We characterize availability provided by replication in terms of the number of replicas maintained in the system. While in principle the number of replicas maintained for each slice can differ, we assume the same number of replicas is used for all the slices. This derives from the fact that we assume that nodes are not associated with individual reliability profiles (Section 2.5). Since all slices are needed to reconstruct the resource, using fewer replicas for any of the slices

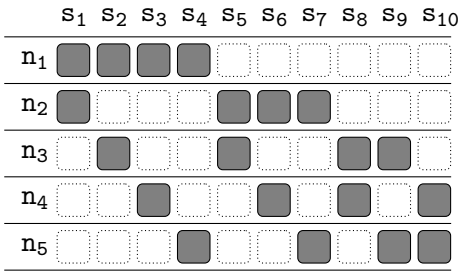


Figure 2.4: An example of a minimal 3-protected and 2-replicated allocation function

would decrease the availability of the resource, which will be dictated by such a lower bound. The following definition formalizes the replication degree of an allocation function.

Definition 2.3.2 (r-Replicated allocation function) *Let $\mathscr{S}$ be a set of slices composing a resource, $\mathscr{N}$ be a set of nodes, and φ be an allocation function. Function φ is* r-replicated *iff* $\forall \mathtt{s_i} \in \mathscr{S}$, $|\varphi(\mathtt{s_i})| \geq r$.

For instance, the allocation function in Figure 2.4 is 2-replicated, as two copies are maintained for each slice.

We characterize the protection offered by an allocation in terms of the minimum number of nodes required to reconstruct a resource, as formalized by the following definition.

Definition 2.3.3 (k-Protected allocation function) *Let $\mathscr{S}$ be a set of slices composing a resource, $\mathscr{N}$ be a set of nodes, and φ be an allocation function. Function φ is* k-protected *iff for each* $\mathtt{N_i} \subset \mathscr{N}$, *with* $|\mathtt{N_i}| \leq k$, $\exists \mathtt{s_j} \in \mathscr{S}$ *s.t.* $\varphi(\mathtt{s_j}) \cap \mathtt{N_i} = \emptyset$.

A k-protected allocation function guarantees distribution of slices to nodes in such a way to dictate the cooperation of no less than $k+1$ nodes to collect all the slices composing the resource (and hence enabling retrieving its plaintext). In other words, a k-protected allocation function guarantees protection of the resource against malicious (i.e., colluding) behavior of up to k nodes. In fact, with a k-protected allocation function, for each coalition of k nodes in $\mathscr{N}$, there is at least a slice that is not stored at any of the nodes in the coalition. Hence, such a coalition can neither decrypt the resource with a brute-force attack, nor prevent its deletion. The allocation function in Figure 2.4 is 3-protected: any subset of 3 out of the 5 nodes misses at least a slice. For instance, coalition $\{\mathtt{n_1},\mathtt{n_2},\mathtt{n_3}\}$ misses slice $\mathtt{s_{10}}$, while coalition $\{\mathtt{n_1},\mathtt{n_2},\mathtt{n_4}\}$ misses slice $\mathtt{s_9}$. On the contrary, the allocation function in Figure 2.5, on the same slices and nodes, is not 3-protected (but only 2-protected): coalition $\{\mathtt{n_1},\mathtt{n_3},\mathtt{n_4}\}$ jointly possesses all the slices.

We refer to an allocation function that is r-replicated, according to Definition 2.3.2, and k-protected, according to Definition 2.3.3, as a (k,r)-*allocation*.

Definition 2.3.4 ((k,r)-Allocation) *Let $\mathscr{S}$ be a set of slices composing a resource, $\mathscr{N}$ be a set of nodes, and φ be an allocation function. Function φ is a* (k,r)-allocation *iff it is* k-protected *and* r-replicated.

According to Definitions 2.3.2 and 2.3.3, a (k,r)-allocation is also a (k',r')-allocation, for any $r' \leq r$ and any $k' \leq k$. In fact, trivially, an allocation function providing r replicas also provides $r' < r$ replicas. Analogously, an allocation function protecting a resource from coalitions of k

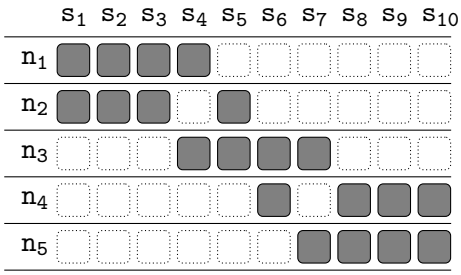


Figure 2.5: An example of 2-replicated allocation function that is not 3-protected

nodes also protects the resource from coalitions of $k' < k$ nodes. Among all (k,r)-allocations, we are interested in identifying those for which k and r represent the highest values satisfying the availability and protection properties (i.e., satisfying the properties in a minimal way). We call such allocation functions minimal, as formalized by the following definition.

Definition 2.3.5 (Minimal (k,r)-allocation) *Let $\mathscr{S}$ be a set of slices composing a resource, $\mathscr{N}$ be a set of nodes, and φ be a (k,r)-allocation. Function φ is* minimal *iff:*

1. *it is not $(k+1)$-protected;*
2. $\forall \mathtt{s_i} \in \mathscr{S}$, $|\varphi(\mathtt{s_i})| = r$.

According to Definition 2.3.5, a *minimal* (k,r)-allocation is an allocation that guarantees protection against coalitions of up to k (but no more) nodes and that uses exactly r replicas. The allocation function in Figure 2.4 is an example of minimal $(3,2)$-allocation. In the following, we will restrict our attention to minimal allocation functions and, when talking about a (k,r)-allocation, we will implicitly assume such minimality.

2.4 Slicing and allocation strategies

In the absence of replication, producing an allocation that guarantees k-protection, that is, a $(k,1)$-allocation, is straightforward: it is sufficient to split the resource into $k+1$ slices and allocate each slice to a different node. When considering replication, different approaches can be taken for allocation, differing in the granularity of slicing and in how allocation diversifies the storage at different nodes. In the following, we discuss these options. In the discussion, in addition to parameters k and r introduced before, we will use parameters s, denoting the number of slices in which a resource is split, and n, denoting the number of nodes to be involved in the allocation of a resource. Different approaches vary in the number s of slices to be considered and in the number n of nodes to be involved for providing a (k,r)-allocation. We note that, with respect to nodes, the only parameter to be considered in the allocation strategies is the number n of nodes to be involved (the specific nodes to be involved can be selected randomly). We identify and study the behavior of two approaches for producing a (k,r)-allocation. The first approach aims to minimize the number of slices (*Min_slices*), while the second aims to minimize the number of nodes (*Min_nodes*). We analyze these two approaches as they represent the two extremes with respect to granularity of slicing and diversification of allocation. Their analysis permits to highlight the characteristics of fine-grained (*Min_nodes*) and coarse-grained (*Min_slices*) slicing, and can also represent a reference for intermediate configurations.

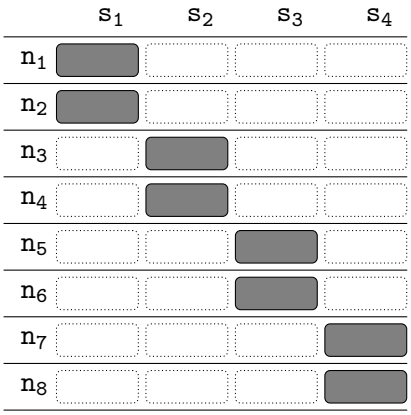


Figure 2.6: An example of $(3,2)$-allocation that minimizes the number of slices

2.4.1 Minimizing the number of slices

We start noting that the number s of slices involved for guaranteeing a (k,r)-allocation must be such that $s \geq k+1$. In fact, there should be at least $k+1$ slices to guarantee k-protection.

A simple approach for determining a (k,r)-allocation extends the natural approach of producing $k+1$ slices, by simply considering their replication at different nodes. Such an approach is characterized by a *coarse-slicing*, since minimizing the number of slices clearly entails a larger size for them, and by *consistent replication* (i.e., nodes have no intersection or complete intersection of stored slices).

We observe that a (k,r)-allocation function using the minimum number ($s = k+1$) of slices implies that:

1. a node maintains at most one slice;
2. the number of nodes involved in the allocation is exactly r times the number of slices (i.e., $n = r \cdot (k+1)$).

The first observation derives from the fact that, since there are only $k+1$ slices, placing more than one slice on a node would imply the existence of a set of k nodes able to reconstruct the resource and therefore would not guarantee k-protection anymore. The second observation naturally derives from the first, considering that every slice needs to be replicated r times.

As an example, a $(3,2)$-allocation using the minimum number of slices would imply splitting the resources into 4 $(= 3+1)$ slices, generating 2 copies of each slice, to be distributed at 8 different nodes. Figure 2.6 illustrates an example of allocation function enforcing this.

A (k,r)-allocation that uses the minimum number of slices $s = k+1$ well resists to failures. Indeed, $k+1$ nodes out of $r \cdot (k+1)$ are sufficient to reconstruct the resource content, as long as one replica of each slice is available. However, the number of nodes used by such an allocation function quickly grows with k and r. For instance, a $(10,5)$-allocation would need 55 $(= 5 \cdot (10+1))$ nodes.

2.4.2 Minimizing the number of nodes

At the other end of the spectrum of possible strategies for defining and distributing slices to guarantee a (k,r)-allocation, there are functions minimizing the number of nodes to be involved in the

distribution (and deriving the number of slices in which the resource needs to be split based on this).

A trivial lower bound on the number of nodes that need to be involved in a (k,r)-allocation is $n \geq max(k+1,r)$, since there should be at least r nodes to hold r replicas and at least $k+1$ nodes to guarantee k-protection. The minimum number of nodes to be involved to guarantee (k,r)-allocation is actually higher than that as it needs to be at least the sum of the protection and replication parameters (k and r).

This derives from two simple observations. First, to guarantee k-protection, for each coalition of k nodes, there must exist at least one slice that is not stored at any of the nodes in the coalition. Second, to provide r-replication, such a slice should be stored at (at least) r nodes that are not in the coalition. As we will illustrate in the following, $k+r$ nodes, besides been necessary, are also sufficient to define a (k,r)-allocation.

While using the minimum number of slices applies a coarse slicing with consistent replication, using the minimum number of nodes applies a *fine-grained slicing* with *diversified replication* across nodes. Intuitively, instead of splitting the resource into slices and allocating to each node a single slice, minimizing the number of nodes requires slicing the resource into more fine-grained slices and allocating the slices to nodes in a diversified manner, to guarantee that no set of k nodes jointly possesses all the slices. The definition of the allocation requires then to identify the number of slices in which a resource needs to be split, which must be sufficient to distribute the r replicas to nodes while ensuring k-protection. The minimum number of slices needed for ensuring that no set of k nodes is able to reconstruct the resource when using $k+r$ nodes, clearly happens when any set of k nodes misses exactly one slice (which, given r-replication, would instead be stored at the r nodes not belonging to the set) and no two coalitions miss the same slice. In fact, if two sets of k nodes miss the same slice, such a slice could not have r replicas when using only $k+r$ nodes. The number of required slices can then be identified as the number of coalitions of k nodes out of $k+r$, that is $\binom{k+r}{k}$.

A (k,r)-allocation that uses $k+r$ nodes and $\binom{k+r}{k}$ slices has two interesting properties. The first one, already noted, is that any coalition of k nodes *misses exactly one* slice. The second one, deriving from the fact that the missing slice is different for different coalitions, is that *any* set of $k+1$ nodes is sufficient to reconstruct the resource (differently from the *Min_slices* approach where at least $k+1$ nodes are needed to reconstruct the resource but not any set of $k+1$ nodes guarantees that).

A (k,r)-allocation that minimizes the number of nodes can be obtained by assuming $\mathcal{N}$ to comprise $k+r$ nodes and proceeding as follows. Let $2^{\mathcal{N}}_k = \{\mathtt{N_i} \in 2^{\mathcal{N}} : |\mathtt{N_i}| = k\}$ be all subsets of k nodes in $\mathcal{N}$. For each slice $\mathtt{s_i} \in \mathcal{S}, i = 1,\ldots,\binom{k+r}{k}$, $\varphi(\mathtt{s_i}) = \{\mathcal{N} \setminus \{\mathtt{N_i}\}$ with $\mathtt{N_i} \in 2^{\mathcal{N}}_k\}$. Intuitively, for each slice $\mathtt{s_i}$, $\varphi(\mathtt{s_i})$ selects a coalition of k nodes that misses $\mathtt{s_i}$ and allocates slice $\mathtt{s_i}$ to all the other nodes. This guarantees that each coalition ($\mathtt{N_i}$) of k nodes misses at least one slice ($\mathtt{s_i}$), providing k-protection. Slice $\mathtt{s_i}$, which represents the missing slice for coalition $\mathtt{N_i}$, is stored at all the other $n-k=r$ nodes in $\mathcal{N}$, providing r-replication. Intuitively, in a (k,r)-allocation using the minimum number of nodes, no two slices are allocated exactly to the same set of nodes (i.e., $\forall \mathtt{s_i}, \mathtt{s_j} \in \mathcal{S}$, $\varphi(\mathtt{s_i}) \neq \varphi(\mathtt{s_j})$). In fact, the possible subsets of r nodes in $\mathcal{N}$ is $\binom{k+r}{r}$ and $\binom{k+r}{k} = \binom{k+r}{r}$.

For example, a $(3,2)$-allocation using the minimum number of nodes requires $n = k+r = 3+2 = 5$ nodes and the use of $s = \binom{k+r}{k} = \binom{5}{3} = 10$ slices. Figure 2.4 illustrates an example of $(3,2)$-allocation distributing 10 slices over 5 nodes. The allocation is a $(3,2)$-allocation since it replicates each slice twice while guaranteeing that no coalition of 3 nodes possesses all the slices.

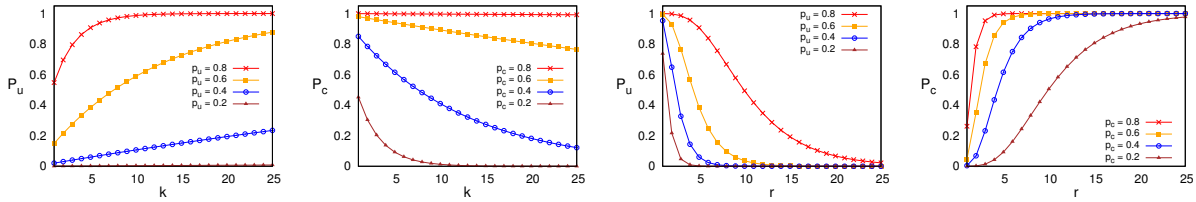
More precisely, any coalition of 3 nodes misses exactly one slice and the missing slice is different for any of such coalitions. For instance, coalition $\{\mathtt{n}_1,\mathtt{n}_2,\mathtt{n}_3\}$ misses slice $\mathtt{s}_{10}$, while coalition $\{\mathtt{n}_1,\mathtt{n}_2,\mathtt{n}_4\}$ misses slice $\mathtt{s}_9$.

2.4.3 Discussion

For simplicity, we have assumed that the data market can arbitrarily split resources as needed for the definition of a (k,r)-allocation. However, thanks to its flexibility, our approach can be adopted also when the encrypted resource is already organized in *chunks* that cannot be split for allocation (e.g., blocks resulting from the AONT algorithm adopted), or in general when slicing is constrained. Indeed, even if in the discussion, for simplicity, we consider slices of equal size, our approach can be adopted also if the size varies. Also, slices can contain non-contiguous chunks of the resource. Clearly, the number of chunks should be sufficient for the definition of a (k,r)-allocation (e.g., $k+1$ and $\binom{n}{k}$ in our two alternative configurations). If the resource includes fewer chunks, it needs to be padded. If the resource includes more chunks than necessary, the data market can combine the chunks in s slices and apply the chosen allocation function over these slices. As an example, to define a (3,2)-allocation for a resource organized in 20 chunks using 5 nodes, chunks can be arbitrarily combined to identify 10 slices for allocation. Alternatively, k-protection and r-replication can be obtained by considering each chunk as a different slice and interpreting the allocation function as periodic in s, or simply by randomly allocating the chunks after the first s (which are the ones necessary to guarantee k-protection). For instance, a (3,2)-allocation for a resource with 20 chunks using 5 nodes can be obtained by applying the allocation function in Figure 2.4 twice (on slices $\mathtt{s}_1,\ldots,\mathtt{s}_{10}$ and $\mathtt{s}_{11},\ldots,\mathtt{s}_{20}$), or by using it for slices $\mathtt{s}_1,\ldots,\mathtt{s}_{10}$ while arbitrarily allocating slices $\mathtt{s}_{11},\ldots,\mathtt{s}_{20}$ at two nodes each.

2.5 Availability and protection guarantees

Parameters r and k introduced in the previous section characterize the degree of replication and of protection against malicious coalitions of nodes. Such parameters provide a clean and precise modeling and allow reasoning about properly setting the number of slices and the number of nodes to be involved in the allocation. The setting of k and r to provide given security and availability guarantees clearly depends on the specific characteristics of the network. For instance, in a stable network a low number of replicas may suffice to provide high availability, while in a highly dynamic and non-resilient network a higher number of replicas should be used to enjoy the same guarantee. In the same vein, actual protection against possible exposure of a resource to malicious coalitions depends on the nature of nodes involved in the allocation. Consistently with these observations, we note that a natural way to express and reason about availability and protection guarantees is the probability of the resource to become unavailable and the probability of a coalition of malicious nodes to jointly possess all the resource slices. In this section, we illustrate how to derive proper r and k settings to be then used for splitting resources into slices and for slices allocation, starting from the aimed guarantee of availability and security expressed in terms of such probabilities. Clearly, the probability of a resource to become unavailable, or exposed to malicious coalitions, depends on the probability of individual nodes to become unavailable or behaving maliciously. We then introduce the probability of a single node to fail, and hence to become unavailable, denoted p_u, and the probability of a node to behave maliciously, and hence to participate in a malicious coalition compromising protection, denoted p_c. We assume, as common



(a) $k = 1,\ldots,25$, $r = 5$ (b) $k = 1,\ldots,25$, $r = 5$ (c) $k = 5$, $r = 1,\ldots,25$ (d) $k = 5$, $r = 1,\ldots,25$

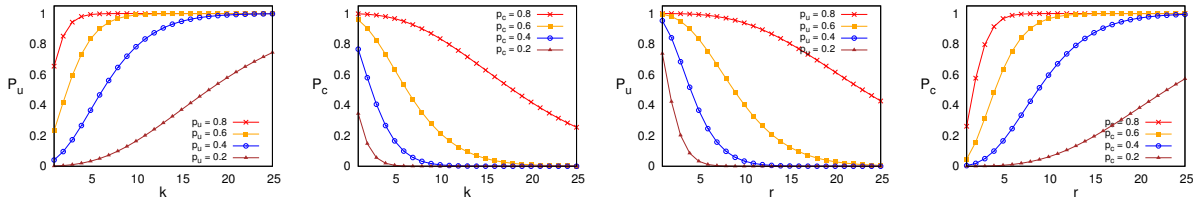
Figure 2.7: Probability that the resource is unavailable (a,c) and that it is exposed (b,d) using a (k,r)-allocation that minimizes the number of slices, with r=5 varying k between 1 and 25 (a,b), and with k=5 varying r between 1 and 25 (c,d)

in decentralized systems, the probability p_u of failure to be the same for all nodes and the failure of any node not to be influenced by the failure of the other nodes. This assumption enables a clean modeling, which can be taken as a reference for reasoning on different probability distributions. Since the selection of storage nodes is driven by a pseudorandom function, we also consider a uniform probability p_c of compromise and assume independence of compromise events on different nodes. We introduce the probability of a resource to become unavailable, denoted P_u, and of being exposed to a malicious coalition, denoted P_c, when using a (k,r)-allocation. The analysis will then guide the identification of the values for k and r to be used to guarantee that P_u and P_c do not exceed a given threshold. We discuss separately the *Min_slices* and *Min_nodes* allocation strategies introduced in the previous section, which, as we will see, exhibit a different behavior with respect to availability and security guarantees.

2.5.1 Min_slices allocation

Using a (k,r)-allocation with the minimum number of slices, unavailability of a resource happens when, for any of the $k+1$ slices composing the resources, all the r nodes storing the replica of the slice fail. The probability of such an event to happen is $P_u = 1-(1-(p_u)^r)^{k+1}$, where $(1-(p_u)^r)$ is the probability that one of the r replicas of a slice is available and, for the assumption on the independence of the failure events, $(1-(p_u)^r)^{k+1}$ is the probability that one replica of each of the $k+1$ slices is available. In the same vein, the resource becomes exposed (and hence a compromise happens and deletion cannot be guaranteed) when a coalition of malicious nodes collectively possesses all the $k+1$ slices, that is, when the coalition contains $k+1$ nodes each possessing a different slice. The probability of such an event to happen is $P_c = (1-(1-p_c)^r)^{k+1}$, where $(1-p_c)^r$ is the probability that one replica is stored on a node that is not part of a coalition and, consequently, $1-(1-p_c)^r$ is the probability that one replica is exposed. Since such an exposure must involve all the $k+1$ slices, the probability that a coalition possesses all the slices is $(1-(1-p_c)^r)^{k+1}$.

Figure 2.7 illustrates how k and r affect the values of P_u (Figure 2.7(a,c)) and P_c (Figure 2.7(b,d)), considering different values of p_u and p_c, respectively. The values considered for p_u and p_c are 0.2, 0.4, 0.6, and 0.8. These values, extremely pessimistic with respect to what can be expected in real systems, have been chosen to study the behavior of the probabilistic formulas. Figure 2.7(a) reports the values of P_u assuming a fixed number $r = 5$ of replicas and varying k between 1 and 25. Figure 2.7(c) reports the values of P_u assuming a fixed $k = 5$ and varying the number r of replicas between 1 and 25. Figures 2.7(b,d) report the values of P_c in the same settings of Figures 2.7(a,c). As it can be seen from Figure 2.7(a), P_u increases as the value of k increases, because the number



(a) $k = 1,\ldots,25, r = 5$ (b) $k = 1,\ldots,25, r = 5$ (c) $k = 5$, $r = 1,\ldots,25$ (d) $k = 5$, $r = 1,\ldots,25$

Figure 2.8: Probability that the resource is unavailable (a,c) and that it is exposed (b,d) using a (k,r)-allocation that minimizes the number of nodes, with $r = 5$ varying k between 1 and 25 (a,b), and with $k = 5$ varying r between 1 and 25 (c,d)

of nodes used in the allocation increases and therefore the probability of availability of a larger number of slices decreases. Indeed, the number of nodes necessary to reconstruct a resource grows with k (it is $k+1$), and the probability of availability of all the nodes necessary to reconstruct the resource decreases. However, P_u remains low if the failure probability of a single node p_u is low. Probability P_u instead decreases as the value of r increases (Figure 2.7(c)), because each slice will be stored on a larger number of nodes, reducing the risk of unavailability. Figure 2.7(b) shows that P_c decreases as k increases because the number of nodes that should be part of a coalition increases, meaning that the probability of forming a coalition decreases. Probability P_c increases as r increases (Figure 2.7(d)), because the number of replicas of each slice increases and therefore also the probability that one replica is stored on a compromised node increases.

2.5.2 Min_nodes allocation

Using a (k,r)-allocation with the minimum number of nodes, the unavailability of the resource occurs when any combination of r (or more) nodes becomes unavailable. In fact, regardless of the slices that those nodes store, such an event causes at least one slice to be unavailable. The probability P_u that a resource becomes unavailable is then $P_u = \sum_{i=r}^{k+r} \binom{k+r}{i} (p_u)^i (1-p_u)^{k+r-i}$, where the binomial coefficient $\binom{k+r}{i}$ is the number of all possible combinations of i nodes over $k+r$, with i varying in the range $r,\ldots,k+r$, that can be unavailable; $(p_u)^i$ is the probability that i nodes are unavailable; and $(1-p_u)^{k+r-i}$ is the probability that the remaining nodes (i.e., $k+r-i$) are available. In the same vein, any coalition of $k+1$ nodes causes an exposure of the resource, regardless of the slices they store. Relying on the minimum number of nodes, in fact, implies that any coalition of $k+1$ nodes possesses all the slices. The probability P_c of a compromise is then $P_c = \sum_{i=k+1}^{k+r} \binom{k+r}{i} (p_c)^i (1-p_c)^{k+r-i}$, where the binomial coefficient $\binom{k+r}{i}$ is the number of all possible coalitions of i nodes over $k+r$ nodes, with i varying in the range $k+1,\ldots,k+r$; $(p_c)^i$ is the probability that i nodes form a coalition; and $(1-p_c)^{k+r-i}$ is the probability that the remaining nodes (i.e., $k+r-i$) are not compromised.

Figure 2.8 illustrates how k and r affect the values of P_u (Figures 2.8(a,c)) and P_c (Figures 2.8(b,d)), considering different values of p_u and p_c, respectively. The values considered for p_u and p_c are 0.2, 0.4, 0.6, and 0.8. Figure 2.8(a) reports the values of P_u assuming a fixed number $r = 5$ of replicas and varying k between 1 and 25. Figure 2.8(c) reports the values of P_u assuming a fixed $k = 5$ and varying the number r of replicas between 1 and 25. Figures 2.8(b,d) report the values of P_c in the same settings as Figures 2.8(a,c). From the figures, it is immediate to see that P_u and P_c present a similar behavior when adopting a configuration minimizing the number of slices

and of nodes (i.e., P_u increases as k grows and decreases as r grows, while P_c decreases as k grows and increases as r grows).

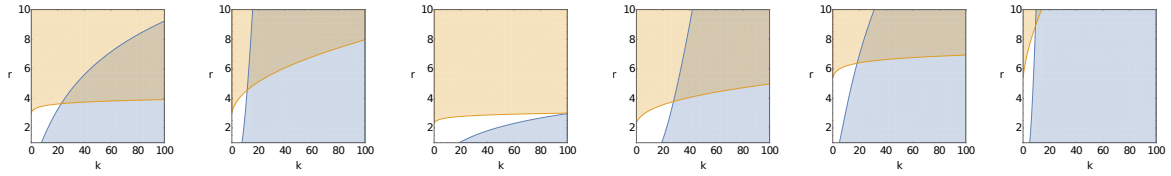
2.5.3 Setting k and r

Our modeling of the probability that a resource is not available (P_u) and that it is exposed (P_c) can be used to set appropriate values for parameters k and r. To this purpose, fixing the maximum threshold P_u^{max} of resource unavailability and P_c^{max} of resource exposure, we compute all the configurations of k and r that guarantee $P_u \leq P_u^{max}$ and $P_c \leq P_c^{max}$. Clearly, the values of k and r for the configurations satisfying the thresholds depend on the chosen allocation function.

Comparing the evolution of the probability P_u that a resource becomes unavailable using the *Min_slices* and *Min_nodes* allocation strategies, varying k (Figure 2.7(a) and Figure 2.8(a)), we can easily see that *Min_slices* is more robust against node failure (i.e., P_u increases slowly) than *Min_nodes*. This is due to the fact that even if the number of nodes involved in the allocation increases in both configurations, with an allocation that minimizes the number of nodes the impact of a node failure on the availability of the resource is significant. A similar comment applies when comparing how P_u evolves in the two configurations varying the number r of replicas (Figure 2.7(c) and Figure 2.8(c)). In this case, the decrease of P_u with *Min_slices* is faster than the decrease of P_u with *Min_nodes*. Therefore, we can conclude that, for configurations with the same values for r and k, *Min_slices* exhibits higher availability.

Comparing the evolution of the probability P_c that a resource is exposed due to a coalition of at least $k+1$ nodes using *Min_slices* and *Min_nodes* allocation strategies, varying k (Figure 2.7(b) and Figure 2.8(b)), we can easily see that *Min_nodes* is more robust (i.e., P_c decreases faster) than *Min_slices*. This is due to the fact that, with an allocation that minimizes the number of nodes, the probability of forming a coalition of at least $k+1$ nodes among the $k+r$ nodes is smaller than the probability of controlling at least one of the r nodes for each of the $k+1$ slices of the allocation that minimizes the slices. A similar comment applies when comparing how P_c evolves in the two configurations varying the number of replicas (Figure 2.7(d) and Figure 2.8(d)). The increase of probability P_c using *Min_slices* is faster than the increase of P_c using *Min_nodes*, because it is more difficult to control at least $k+1$ of the $k+r$ nodes than to control at least one node in each of the distinct $k+1$ groups of r nodes. Therefore, we can conclude that, for configurations with the same values for r and k, *Min_nodes* exhibits higher security.

When the allocation function has been chosen, given the maximum threshold P_u^{max} of resource unavailability and P_c^{max} of resource exposure, different configurations of k and r guarantee that $P_u \leq P_u^{max}$ and $P_c \leq P_c^{max}$. Among all these configurations, the ones with low replication factor (r) require less storage and have lower economic costs, while the ones involving a limited number (n) of nodes enjoy simplicity in the management of the system and better performance of access operations (less connections have to be established). Figure 2.9 considers three different network configurations, characterized by a different probability p_u for single nodes to fail and a different probability p_c to behave maliciously, and illustrates the configurations of k and r satisfying the above thresholds using *Min_slices* and *Min_nodes* allocation strategies. In the figure, the orange area on the top-left represents the configurations of k and r that satisfy the availability requirement (i.e., $P_u \leq 10^{-7}$), while the blue area on the bottom-right represents the configurations that satisfy the security requirement (i.e., $P_c \leq 10^{-6}$). The intersection between the orange and blue areas represents configurations that provide both availability and security guarantees. Among these configurations, the one located on the left/bottom corner of the intersecting area is the one to be



(a) *Min_slices*, p_u=0.005, p_c=0.2 (b) *Min_nodes*, p_u=0.005, p_c=0.2 (c) *Min_slices*, p_u=0.001, p_c=0.5 (d) *Min_nodes*, p_u=0.001, p_c=0.5 (e) *Min_slices*, p_u=0.05, p_c=0.1 (f) *Min_nodes*, p_u=0.05, p_c=0.1

Figure 2.9: *Min_slices* and *Min_nodes* (k,r)-allocations that guarantee $P_u \leq 10^{-7}$ and $P_c \leq 10^{-6}$ with different values for p_u and p_c

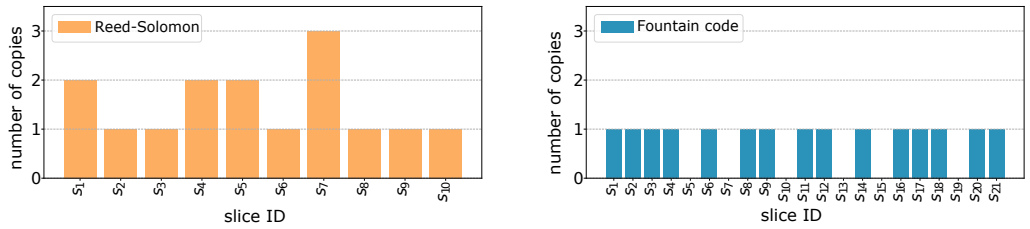
preferred as the number of nodes and replicas is minimum.

Figures 2.9(a,b) consider nodes with $p_u = 0.005$ and $p_c = 0.2$. The optimal configuration for *Min_slices* it is $k = 26$ and $r = 4$ (i.e., $n = 108$), while for *Min_nodes* allocation is $k = 12$ and $r = 5$ (i.e., $n = 17$). The second allocation, although more expensive on storage, due to one additional replica, considerably reduces the number of nodes involved in the storage of the resource compared to the adoption of the first allocation function. Our analysis demonstrates that this is a general behavior: *Min_nodes* requires the same (or a slightly higher) number r of replicas and a significantly lower number n of nodes than *Min_slices*. This observation is confirmed by the extreme scenarios illustrated in Figures 2.9(c,d), considering highly reliable ($p_u = 0.001$) but lowly trusted ($p_c = 0.5$) nodes, and in Figures 2.9(e,f), considering unreliable ($p_u = 0.05$) but relatively trusted ($p_c = 0.1$) nodes. The optimal configurations in Figures 2.9(c,d) are $k = 100$ and $r = 3$ for *Min_slices* (i.e., $n = 303$), and $k = 27$ and $r = 4$ for *Min_nodes* (i.e., $n = 31$, meaning that the number of nodes is ten times smaller). The optimal configurations in Figures 2.9(e,f) are $k = 10$ and $r = 9$ (i.e., $n = 99$) for *Min_slices*, and $k = 18$ and $r = 7$ (i.e., $n = 25$) for *Min_nodes*.

Our analysis confirms that, for a wide range of values for P_u and P_c and assumptions on the node availability p_u and compromise risk p_c, our approach is able to identify a configuration of r and k with manageable complexity (i.e., a reasonable number of replicas and of nodes). We note that, even when r and k grow, the minimum number of slices composing a resource remains limited.

2.6 Encoding strategy for improving availability

Typically, DCS networks provide availability by employing slice replication. Instead of simple replication in the allocation, several DCS networks leverage the dynamic application of erasure codes (e.g., Reed-Solomon). With Reed-Solomon, if a node participating in the service becomes unavailable, its slices are dynamically re-allocated to another node. The advantage of Reed-Solomon with respect to simple replication is that it provides reliability guarantees at a fraction of the storage overhead that would come with replication. However, it has two main drawbacks. First, it is a fixed-rate encoding technique and therefore computing the slice to be re-allocated requires reconstructing the complete resource. Second, if the old node originally storing a (then re-allocated) slice comes back online, more replicas than actually needed would be available for the same slice and the economic cost brought by the involvement of more nodes does not bring a clear advantage for availability. To overcome these drawbacks, we propose to adopt fountain codes, in contrast to traditional erasure codes like Reed-Solomon, for computing slices to be distributed to nodes in a DCS. Our approach avoids potentially excessive generation of slices which may turn out to be unnecessary when – as it is often the case – nodes unavailability is only tempo-



(a) Reed-Solomon

(b) Fountain codes

Figure 2.10: An example of slice generation/replication using Reed-Solomon (a) and fountain codes (b) in a dynamic scenario

rary.

Given a resource, encrypted using AONT to protect confidentiality, the ciphertext is encoded using fountain codes to provide availability. The ciphertext is organized in f original fragments and encoded into a set $\mathscr{S} = \{\mathtt{s}_1, \ldots, \mathtt{s}_\mathtt{s}\}$ of $\mathtt{s}$ slices allocated to $\mathtt{s}$ randomly chosen nodes $\mathscr{N} = \{\mathtt{n}_1, \ldots, \mathtt{n}_\mathtt{s}\}$ (i.e., each slice is allocated to a different node). To retrieve the resource content, it is sufficient to contact an arbitrary subset of f nodes and download their slices, which are then combined to reconstruct the resource.

Since fountain codes are rateless, if a node $\mathtt{n}_\mathtt{i}$ leaves the DCS, the data market does not need (as it would happen using Reed-Solomon) to reconstruct its slice $\mathtt{s}_\mathtt{i}$ and re-allocate it to a different node to guarantee the same resource availability. Indeed, it is sufficient to generate a new slice $\mathtt{s}_{\mathtt{s}+1}$ (different from $\mathtt{s}_1, \ldots, \mathtt{s}_\mathtt{s}$) and allocate it to a new node $\mathtt{n}_{\mathtt{s}+1}$. After the generation and allocation of $\mathtt{s}_{\mathtt{s}+1}$, any subject who is authorized for the resource can still reconstruct the resource content by downloading any subset of f slices from the resulting set $(\mathscr{S} \setminus \{\mathtt{s}_\mathtt{i}\}) \cup \{\mathtt{s}_{\mathtt{s}+1}\}$ of $\mathtt{s}$ slices. The generation of a new slice, in contrast to the replication of the unavailable one, provides higher resource availability. Indeed, every slice is unique and equally contributes to the reconstruction of the resource. Hence, when the previously unavailable node $\mathtt{n}_\mathtt{i}$ comes back online, the increased economic cost due to the involvement of a larger number $(\mathtt{s}+1)$ of nodes comes with an actual advantage in terms of higher availability. In fact, there will be $\mathtt{s}+1$ (instead of $\mathtt{s}$) different slices stored at $\mathtt{s}+1$ different nodes that can all be used for resource reconstruction. Note that the generation of a new slice causes a limited overhead since it implies the download of a subset of the slices in $\mathscr{S}$, without the need to reconstruct the resource (as required by other erasure codes).

As discussed, the rateless characteristic of fountain codes allows the adaptive adjustment of the number of slices available for reconstructing a resource, thus impacting the availability and security guarantees. We use this characteristic (see Section 2.7) in such a way that, when the availability guarantees go below a given threshold, the data market can generate a new slice. Analogously, when the risk of confidentiality exposure is above a given threshold due to the presence of a high number of slices, the data market can ask the data owner to re-encrypt the resource and generate a new set of $\mathtt{s}$ slices.

Figure 2.10 illustrates an example that compares the set of slices in a DCS when using Reed-Solomon and fountain codes, assuming the same number of nodes that leave/join the DCS. Here, we assume that the data market partitions the resource in f=7 fragments and encodes it in $\mathtt{s}$=10 slices and that nodes where the slices are stored leave and then possibly re-join the network. Since Reed-Solomon reacts to a node failure replicating the slice of the failed node, the set $\mathscr{S}$ of slices representing the resource never changes but the number of copies grows (e.g., $\mathtt{s}_7$ has three copies).

Fountain codes do not cause any replication, but a new slice is generated at each failure. Note that the two techniques imply the same economic cost for the data market, since each failure causes the allocation of a (new or duplicated) slice to a node. In the considered example, the market pays for 15 slices in both scenarios.

2.7 Availability and security guarantees with encoding

Node failure has an impact on both resource availability and confidentiality. Indeed, when one of the nodes $\mathtt{n_i}$ fails, the encoded slice $\mathtt{s_i}$ it stores cannot be used to reconstruct the resource content. Hence, to reconstruct the resource, it is necessary to retrieve f out of $\mathtt{s}-1$, in contrast to $\mathtt{s}$, slices. When $\mathtt{n_i}$ re-joins the DCS, it still stores $\mathtt{s_i}$ and this could have a positive effect on resource availability, since the slice at the node could be used to reconstruct the resource. However, node $\mathtt{n_i}$ re-joining the DCS could have a negative impact on security, since $\mathtt{n_i}$ could exploit its knowledge of $\mathtt{s_i}$ and collude with other $f-1$ nodes to reconstruct the resource (or prevent its deletion). Similarly, the generation and allocation of a new slice $\mathtt{s_{s+1}}$ improves resource availability, but also naturally reduces security since there is a higher number of nodes that could possibly collude.

In this section, we analyze the advantages provided by fountain codes on availability guarantees, by studying the probability P_u that a resource becomes unavailable as a consequence of nodes leaving and re-joining the network. We also evaluate the risks that the adoption of our solution causes in terms of security, by studying the probability P_c that a coalition of malicious nodes has enough slices to compromise resource security. These probabilities depend on the probability p_u that a node fails, and on the probability p_c that a node is malicious and interested in colluding with other nodes to breach the confidentiality of the resource. For simplicity, we assume p_u and p_c to be the same for all the nodes.

2.7.1 Availability guarantees

When using $\mathtt{s}$ slices to encode a resource split into f original fragments, the resource becomes unavailable when more than $\mathtt{s}-f$ nodes fail (i.e., when less than f slices can be accessed). The probability of such an event to happen is $P_u{=}\sum_{i=\mathtt{s}-f+1}^{\mathtt{s}} \binom{\mathtt{s}}{i} {p_u}^i (1-p_u)^{\mathtt{s}-i}$. Probability P_u increases when one of the nodes fails (or leaves the DCS) since the slice it stores is no more available, while it decreases any time a new slice is generated or a failed node re-joins the network.

Even if, in principle, the data market should react every time a node $\mathtt{n_i}$ fails by generating a new slice, this practice is expensive and may not even be necessary (e.g., if $\mathtt{n_i}$ re-joins the network in a few hours). The increase in P_u caused by the failure of a node may not be critical in a scenario where nodes dynamically leave and re-join the system frequently. Indeed, the reduction in P_u may be temporary and may not considerably affect the possibility to reconstruct the resource. To properly take into consideration these aspects, we propose a solution where the data market takes corrective actions (i.e., create a new slice) only when resource availability is considered at risk.

Our solution is based on the definition of two thresholds for P_u, P_u^{min} and P_u^{max}, identifying the range of values considered acceptable by the data market for guaranteeing the availability of the resource. Intuitively, these thresholds influence the maximum and minimum number of available slices in the system and represent:

- P_u^{max}: the maximum probability of failure that the data market can tolerate, which corresponds to the minimum number of slices ($\geq f$) the market considers desirable to keep resource unavailability under control;

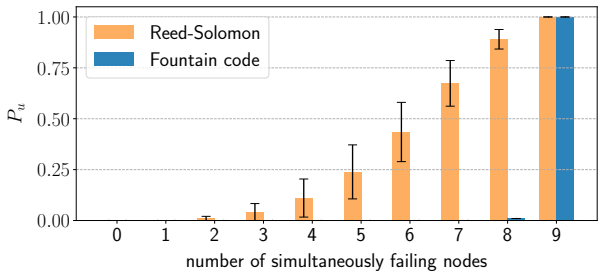


Figure 2.11: Probability that a resource becomes unavailable using Reed-Solomon and fountain codes, assuming p_u=0.015, P_u^{min}=10^{-12}, P_u^{max}=10^{-5}, f=7 fragments, and $\mathtt{s}$ in [10,15]

- P_u^{min}: the minimum probability of failure fixed by the data market based on its economic availability, which corresponds to the maximum number of slices the market can afford.

The data market does not react every time a node leaves or re-joins the DCS, but only if this event causes P_u to become higher than P_u^{max} or smaller than P_u^{min}. If P_u exceeds P_u^{max}, the data market generates a new slice and allocates it to a new node. If P_u goes below P_u^{min}, the data market can terminate the contract with one of the nodes in the system (e.g., with the one that has been off-line the most) and stop paying for its services.

Consider, as an example, a resource partitioned in f=7 original fragments and encoded in $\mathtt{s}$=10 slices allocated at nodes with probability p_u=0.015 of failure. Figure 2.11 compares the probability P_u of unavailability of a resource when using Reed-Solomon and fountain codes, assuming P_u^{min}=10^{-12} and P_u^{max}=10^{-5}, and varying the number of nodes that the data market identifies as unavailable between 0 and 9. The error bars in the figure represent the standard deviation. Note that P_u is initially the same for Reed-Solomon and fountain codes, and evolves in the same manner when nodes leave the DCS. The evolution of P_u when a node re-joins the DCS is instead considerably different: with Reed-Solomon it causes duplication of a slice, with fountain codes it implies the availability of an additional (different) slice. As visible from the figure, the probability that the resource becomes unavailable is higher using Reed-Solomon than using fountain codes. In fact, the nodes storing slices available in a single copy (e.g., $\mathtt{s}_2$ in Figure 2.10(a)) play a critical role in resource reconstruction. Indeed, when failing, they cannot be immediately substituted. With fountain codes, all nodes are equally critical since they all store different slices that can be interchangeably used to reconstruct the resource.

The top chart in Figure 2.12 illustrates an example of evolution of P_u, assuming the adoption of fountain codes, considering the corrective actions taken by the data market. The value of P_u grows every time a node leaves and decreases every time a node re-joins the network. As long as P_u is between P_u^{min} and P_u^{max} (the two red dashed lines in the figure), the data market does not react to node leave and re-join events. In the example, we set P_u^{min} and P_u^{max} in such a way to tolerate the leave and re-join of one node at a time. When P_u reaches P_u^{max} as a consequence of the failure of a node (red triangles 1, 2, and 4 in the figure, indicating unavailability of two nodes), the data market generates and allocates a new slice. The generation of a new slice reduces P_u to a value below P_u^{max} (green circle 1, 2, and 4 in the figure). When P_u reaches P_u^{min} as a consequence of node re-join (red triangle 3 in the figure, indicating re-join of all the three nodes that failed), the data market closes the contract with one of the nodes and P_u returns above the threshold (green circle 3 in the figure).

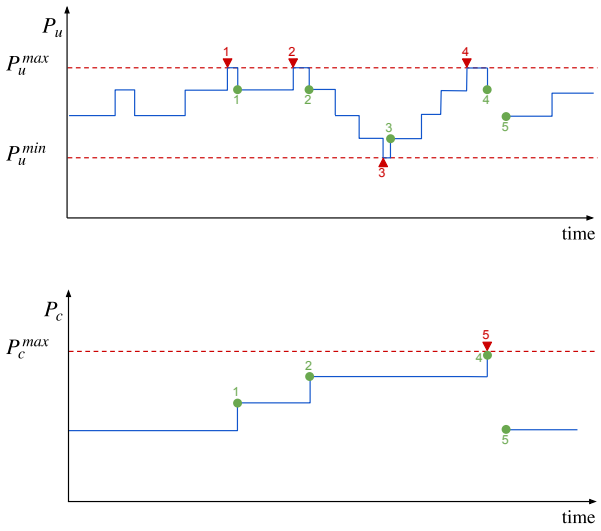


Figure 2.12: An example of evolution of P_u (top chart) and P_c (bottom chart) as a consequence of nodes leaving and re-joining the DCS and of actions taken by the data market

In the next section, we will illustrate that P_u decreases not only when the data market creates a new slice, but also when the resource is re-encrypted.

2.7.2 Security guarantees against malicious coalitions

Since f slices are sufficient to reconstruct the resource, any coalition of at least f malicious nodes can reconstruct the encrypted content of the resource (and its plaintext representation if the key is exposed) and/or prevent its deletion by retaining their local copy of the slice. The probability that f (or more) nodes collude is $P_c = \sum_{i=f}^{s} \binom{s}{i} {p_c}^i (1-p_c)^{s-i}$. The probability P_c that f nodes collude increases whenever the data market generates a new slice and allocates it to a new node. On the contrary, P_c never decreases. Indeed, we cannot assume that nodes leaving the system will not re-join in the future and that they do not have a copy of the slice initially allocated to them, even in case the data market closed the contract. A malicious node can keep a copy of the data on purpose, to prevent resource deletion and possibly sell the slice to non-authorized subjects.

The data market can reduce P_c only by asking the resource owner to re-encrypt the resource with a new key. This process is however quite expensive as it requires to locally reconstruct the resource (downloading f slices), decrypt it, re-encrypt its plaintext content with a new encryption key, encode the resulting ciphertext, and distribute the new set of s slices to s nodes. To limit such overhead, our solution is based on the definition of a threshold P_c^{max} for P_c, representing the maximum probability that the data market (and resource owner) can tolerate that the resource is exposed. When, as a consequence of the generation of a slice, P_c exceeds P_c^{max}, the data market can require the resource owner to re-encrypt the resource. Clearly, slices generated before resource re-encryption cannot be used together with slices generated after re-encryption for resource reconstruction, hence nullifying possible misbehaviors by nodes whose contract has been closed before re-encryption.

The bottom chart in Figure 2.12 illustrates the evolution of P_c due to corrective actions taken by the data market for keeping P_u below P_u^{max} (i.e., the generation of new slices). As long as P_c remains below threshold P_c^{max}, no corrective action is taken (green circles 1 and 2 in the figure). In the example we set P_c^{max} to tolerate the generation of two slices. When the generation of a new

slice causes P_c to reach P_c^{max} (red triangle 5 and green circle 4 in the figure), the market requires the resource owner to re-encrypts the resource. Resource re-encryption resets both P_u and P_c to their initial value (green circle 5 in the figure). The lower is P_u^{max}, the more frequently slices will be generated and the more frequently the resource owner will need to re-encrypt the resource.

2.8 Summary

This chapter focused on the problem of resource storage in the data market and considered the opportunity of relying on DCS services. The approach illustrated in this chapter enables the data market to protect resources and to control their decentralized allocation to different nodes in the network. This chapter investigated different strategies for splitting and distributing resources, analyzing their characteristics in terms of availability and security guarantees. Also, it provided a modeling of the problem enabling owners to control the granularity of slicing and diversification of allocation to ensure the aimed availability and security guarantees.

MOSAICrOWN is now studying an AONT encryption mode enhancing Mix&Slice technique to improve its performance, by reducing the number of mixing phases necessary to encrypt a macro-block.

3. Data wrapping for controlled sharing

In this chapter, we address the problem of enabling controlled data sharing in the data market. In particular, we consider the key requirement of enabling data owners to remain in control of their data, granting selective access to requesting processors (i.e., entities that are interested in accessing portions of the data made available by owners) without the need of delegating the management of access requests to the data market itself. Our solution is based on the controlled adoption of owner-side encryption, by means of which data are wrapped, before being outsourced to the market, with an encryption layer. To enable selective access, different data items are encrypted with different encryption keys, which are made known to authorized subjects. To reduce the overhead of key management, we leverage key derivation techniques. We also consider the possibility to provide incentives to data owners for making their data available in the market, and identify and limit possible misbehaviors occurring with (malicious) entities when dealing with such incentives.

The remainder of this chapter is organized as follows. Section 3.1 discusses the state of the art and highlights MOSAICrOWN innovations. Section 3.2 introduces the reference scenario along with its specific requirements. Section 3.3 discusses some needed preliminaries and the building blocks over which our solution builds. Section 3.4 illustrates our proposal for protecting resources and granting selective access. Section 3.5 presents our approach for managing incentives to data owners and counteracting possible misbehaviors. Section 3.6 discusses some relevant features of our solution, and Section 3.7 concludes the chapter.

3.1 State of the art and MOSAICrOWN innovation

In this section, we illustrate the state of the art and the innovations produced by MOSAICrOWN for controlled sharing of data stored in the market, ensuring that owners remain in control of their data.

3.1.1 State of the art

The enforcement of access restrictions to resources outsourced to external storage and management platforms has been extensively studied in the literature. Recent approaches have explored the adoption of selective owner-side encryption (e.g., [ABFF09, DFJ+10, BDF+16]), wrapping resources with a layer of encryption before outsourcing and distributing keys to authorized subjects, in such a way that all and only authorized subjects can decrypt resources. Such an approach relieves the need of having a trusted party in charge of managing access requests and grant/deny access based on the authorization policy. None of the approaches in this line of work consider the peculiarities of the data market scenario, including the possibility of rewarding owners.

Recently, the adoption of distributed ledgers (such as blockchain) and of smart contracts has been investigated for exchanging resources and/or enforcing access control (e.g., [DEF18,DMR17, SBHD17, KAS+18, DPNH20, ZNP15]). The proposal in [KAS+18] puts forward a solution for

auditable sharing of private data on blockchains, assuming a (collective) authority for enforcing access restrictions. The proposal in [SBHD17] aims at protecting resources through the use of encryption, without considering incentives to data owners and possible misbehaviors that can arise in this context. The problem of trading data has been investigated in [DPNH20], which however adopts Bitcoin transactions and does not support fine-grained authorizations. The approach in [DEF18] focuses on ensuring fairness of resource exchange, without explicit reference to data markets and their need for fine-grained authorizations. A blockchain-based access control model has been proposed in [DMR17], which enables a processor to transfer her access rights to another processor, without ensuring incentives to the owners. The proposal in [ZNP15] protects data using encryption and on-chain storage of pointers to data.

3.1.2 MOSAICrOWN innovation

MOSAICrOWN produced several advancements over the state of the art, which are discussed in this section.

- The first innovation is represented by a technique enabling the enforcement of authorization policies regulating access to resources in the data market scenario, based on specific adaptations of selective encryption and key derivation strategies, considering the possibility of granting incentives to owners for making their resources available on the market.
- The second innovation relates to the identification and characterization of possible misbehaviors that can arise when dealing with incentives to data owners, and the interacting subjects that do not fully trust each other.
- The third innovation is represented by the definition of a solution for counteracting the misbehaviors that might arise in the considered scenario and for incentivizing all parties to behave correctly, based on the combined adoption of blockchain and smart contracts, and on a simple auditing protocol for identifying the subject(s) who misbehaved.

The results obtained by MOSAICrOWN and illustrated in this chapter have been published in [DFLS19].

3.2 Scenario and requirements

We address the problem of allowing subjects to leverage the availability of data market platforms for selectively sharing their data with interested processors (i.e., entities that need to perform some processing on them). Our scenario is characterized by a set $\mathscr{O}=\{o_1,\ldots,o_z\}$ of data owners on one side, and a set $\mathscr{P}=\{p_1,\ldots,p_m\}$ of processors on the other side, who interact through the market platform to trade data, modeled as a generic set $\mathscr{R}=\{r_1,\ldots,r_n\}$ of resources. Such a scenario entails a number of challenges and requirements that need to be carefully addressed. In particular, we investigate two main issues characterizing the data market scenario: *i)* ensuring adequate data protection, and *ii)* ensuring that owners and processors can profitably leverage data market platforms. Our first goal aims at maintaining the owner in control of her data, selectively granting access to resources according to her needs and wishes. As for the second goal, we consider and manage incentives that encourage owners to contribute with their data. While several incentives can nicely fit the data market scenario, in this work we consider incentives based on monetization, according to which a data owner granting access to one of her data items obtains a reward in terms

Request no.	Processor	Requested Resources
1	w	a,b,c
2	x	c,d,e,f
3	y	a,b,c
4	z	b,c
5	z	a,d,e,f
6	w	d,e,f

Figure 3.1: An example of a sequence of six access requests posed by four processors for subsets of six resources

of a payment. Hence, due to the involvement of remuneration incentives and since data owners and processors might not fully trust each other, our second goal also requires to define mechanisms for counteracting possible misbehaviors (e.g., a malicious owner does not provide access to a resource for which she received an incentive or, conversely, a processor claims her money back by maliciously declaring the owner did not grant access despite the provided incentive, see Section 3.5.2 form more details). In designing our solution, we keep the following requirements in mind:

R1. the content of published resources must remain protected, and only authorized processors can access their content;

R2. the data owner must be aware of which processors have access to which resources;

R3. the owner cannot claim that an incentive has not been received while it actually has;

R4. after the owner has granted to a processor access to a resource, the processor cannot claim that access has not been granted (and ask to be refunded the paid incentive).

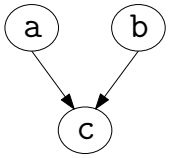
While the first two requirements deal with ensuring data protection, the latter two reduce the possibility of misbehaviors from both the data owner and the processors.

In the remainder of this chapter, we refer our examples to a set $\mathscr{R}=\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e},\mathtt{f}\}$ of six resources uploaded on the market platform. Such resources are of interest for four processors $\mathscr{P}=\{\mathtt{w},\mathtt{x},\mathtt{y},\mathtt{z}\}$, which over time buy access to resources according to the sequence in Figure 3.1.

3.3 Preliminaries

Our solution combines four building blocks: *i)* selective owner-side encryption; *ii)* key derivation; *iii)* blockchain; and *iv)* smart contracts.

Selective owner-side encryption. Selective owner-side encryption consists in encrypting, at the owner side, different resources with different keys, and in distributing keys to processors in such a way that each processor can decrypt all and only the resources she is authorized to access [DFJ+10, DFLS16]. Since the encryption layer is provided at the owner side, resources self-enforce the access restrictions defined over them and their content is protected also to the eyes of the data market. The data owner can then make her resources available to the market platform (which can be hosted, for instance, on the cloud), with the guarantee that only processors knowing the encryption keys will be able to decrypt resources. A straightforward solution to enforce access restrictions through selective encryption consists in encrypting each resource with a different key,



$\mathscr{T}$

From	To	Value
a	c	$k_c \oplus h(k_a, l_c)$
b	c	$k_c \oplus h(k_b, l_c)$

Figure 3.2: An example of key derivation structure and token catalog

and in distributing to each processor the keys of the resources in her capability list. However, this practice would imply a considerable key management burden for processors, who would need to manage one key for each resource for which they are authorized. To mitigate such overhead, we adopt key derivation, as follows.

Key derivation. Key derivation permits to derive the value of an encryption key k_y from the knowledge of another encryption key k_x and of a public label l_y (i.e., a piece of information) associated with k_y [ABFF09, DFJ+10]. The derivation of k_y from k_x is enabled by a public token $t_{x,y}$ computed as $k_y \oplus h(k_x, l_y)$, with $\oplus$ the bitwise `xor` operator, and h a deterministic non-invertible cryptographic function. The derivation relationship between keys can be *direct*, via a single token, or *indirect*, through a chain of tokens. Key derivation structures can be graphically represented as directed acyclic graphs, where vertices represent encryption keys (and their labels), and edges represent tokens. Tokens are physically stored in a public catalog $\mathscr{T}$. Figure 3.2 illustrates an example of derivation among three keys k_a, k_b, and k_c, and the corresponding token catalog $\mathscr{T}$. For simplicity, in our examples we use x to denote the label of key k_x, and use the label of a key to denote the corresponding vertex (e.g., vertex a in Figure 3.2 represents key k_a and its label). In the following, when clear from the context, we will use the terms keys and vertices (tokens and edges, respectively) interchangeably.

Blockchain. A blockchain is a shared and trusted public ledger of transactions, maintained in a distributed way by a decentralized network of peers. Transactions are organized in a list of blocks, linked in chronological order, where each block contains a certain number of transaction records and a cryptographic hash of the previous one. Each transaction is validated by the network of peers, and is included in a block through a consensus protocol. The state of a blockchain is continuously agreed upon by the network of peers: everyone can inspect a blockchain, but no single user can tamper with it, since modifications to the content of a blockchain requires mutual agreement. Once a block is committed, nobody can modify it: updates are reflected in a new block containing the new information. This permits to trust the content and the status of a blockchain, while not trusting the single peers.

Smart contracts. Smart contracts are a powerful tool for establishing contracts among multiple, possibly distrusting, parties. A smart contract is a software running on top of a blockchain and defines a set of rules, on which the interacting parties agree. It can be seen as a set of 'if-then' instructions, defining triggering conditions and subsequent actions capturing and formalizing the clauses of a contract to be signed by the parties. The execution of a smart contract can be trusted for correctness thanks to the underlying blockchain consensus protocols, meaning that all the conditions of the agreement modeled by the contract are certainly met and validated by the network. However, smart contracts and their execution lack confidentiality and privacy, as plain visibility over the content of a contract and over the data it manipulates is necessary for validation [CZK+19].

With reference to the requirements illustrated in Section 3.2, we leverage selective encryption and key derivation to satisfy *R1* and hence for protecting resources granting access according to an authorization policy defined and controlled by the data owner (Section 3.4). In this way, the authorization policy is automatically enforced by wrapping a layer of encryption on resources before being published and distributing encryption keys to authorized users only. We then leverage Blockchain and smart contracts to satisfy *R2* and hence for managing the interactions between processors and the data owner, as well as access requests, by keeping an un-modifiable and undeniable record of the granted access rights (Section 3.5). Finally, we satisfy *R3* and *R4* by counteracting misbehaviors through an audit process that incentivizes all parties to behave correctly.

3.4 Protecting resources

In this section, we illustrate how to enforce requirement *R1* to guarantee that the content of each resource is visible only to authorized processors. For simplicity, but without loss of generality, we consider the set of resources published by one owner o, with the note that the same reasoning applies to each data owner operating on the data market. In line with our assumption of remunerative incentives, we assume for simplicity but without loss of generality that the data owner does not pose access restrictions to her data except from the fact of receiving a payment. Our proposal can however be easily extended to consider additional access conditions.

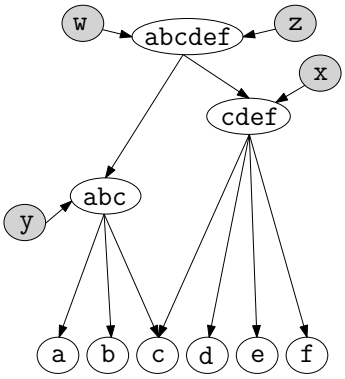
3.4.1 Authorization policy and key derivation structure

We represent the authorization policy $\mathscr{A}$ by means of the *capability lists* of the processors in $\mathscr{P}$, where $\mathsf{cap}(p)$ represents the set of resources for which processor $p \in \mathscr{P}$ is authorized (and, possibly, for which the owner o has received an incentive from p). Every time processor p is granted access to a resource $r \in \mathscr{R}$ (and, as illustrated in the remainder of this chapter, has paid the incentive to the owner), r will be added to her capability list (i.e., $\mathsf{cap}(p) = \mathsf{cap}(p) \cup \{r\}$).

To allow fine-grained access control as demanded by our scenario, without the intervention of the data owner to mediate each access request and without the need to trust the data market to properly enforce access privileges, we leverage selective encryption [DFJ+10]. Since encrypted resources self-enforce the access restrictions defined over them (see Section 3.3), the data owner can physically outsource her published resources to the market platform (which can be hosted, for instance, on the cloud), with the guarantee that only processors provided with the encryption keys will be able to decrypt resources. A straightforward solution to enforce access restrictions through selective encryption consists in encrypting each resource with a different key, and in distributing to each processor the keys of the resources in her capability list. However, this practice would imply a considerable key management burden for processors. To mitigate such overhead, we adopt key derivation (see Section 3.3). Intuitively, each processor p agrees a key k_p with the data owner, who publishes a token allowing p to compute k_r from k_p for each r in $\mathsf{cap}(p)$ (if a same processor can access resources from different owners, she can create a set of tokens enabling her to compute the key shared with each owner from a single (secret) master key). While effective, this simple solution could not be efficient, for instance since it might create more tokens than necessary. A possible optimization is to keep the number of tokens under control and, to this end, the key derivation structure is typically enriched with additional vertices whose key is used for derivation only [DFJ+10, DFLS16]. While in cloud-based scenarios such additional vertices are typically associated with groups of users, the considered scenario would benefit from the definition of keys

Processor	Capability List
w	a, b, c, d, e, f
x	c, d, e, f
y	a, b, c
z	a, b, c, d, e, f

(a)



(b)

Figure 3.3: An example of authorization policy (a), and of a key derivation structure enforcing it (b)

associated with groups of resources. Indeed, such an approach guarantees that each resource r has a different encryption key (the one corresponding to singleton set $\{r\}$). Also, the data market entails a dynamic scenario, with processors joining the system accessing (and possibly paying incentives for) subsets of resources, and it is therefore natural to think in terms of groups of resources in contrast to groups of users.

Formally, a key derivation structure is defined as follows.

Definition 3.4.1 (Key derivation structure) *Given a set $\mathscr{R}=\{r_1,\ldots,r_n\}$ of resources and a set $\mathscr{P}=\{p_1,\ldots,p_m\}$ of processors, a* key derivation structure *over $\mathscr{R}$ and $\mathscr{P}$ is a directed acyclic graph G(V,E) such that:*

1. $\forall v_x \in V$, $(x \in \mathscr{P}) \vee (x \subseteq \mathscr{R})$;
2. $\forall p \in \mathscr{P}$, $v_p \in V$ *and* $\forall r \in \mathscr{R}$, $v_r \in V$;
3. $\forall (v_x, v_y) \in E : (y \subset x) \vee (x \in \mathscr{P} \wedge y \subseteq \mathscr{R})$.

According to the definition above, vertices in the key derivation structure represent processors or sets of resources (Condition 1). Also, the derivation structure has a vertex for each processor and for each resource (Condition 2). Vertices representing processors have only outgoing edges ending at vertices representing (sets of) resources, while edges connecting sets of resources satisfy the subset containment relationship, that is, each vertex is connected to vertices representing subsets of its resources (Condition 3). Each processor p knows the key of its vertex v_p and each resource r is encrypted with the key of its vertex v_r. With reference to our running example defined over six resources and four processors, Figure 3.3(b) illustrates an example of key derivation structure with a vertex for each processor (gray), a vertex for each resource, and additional vertices for subsets of resources. As already noted, for simplicity in the figures we denote each vertex v_x with x (e.g., a is the vertex for resource a and x is the vertex of processor x.)

A key derivation structure correctly enforces an authorization policy $\mathscr{A}$ iff it allows each processor to derive all and only the keys used to encrypt the resources that she is authorized to access, meaning that each processor must be able to reach, starting from its vertex, all and only the vertices representing the resources in her capability list.

Definition 3.4.2 (Correctness) *Given an authorization policy $\mathscr{A}$ over a set $\mathscr{R}$ of resources and a set $\mathscr{P}$ of processors, a key derivation structure G(V,E) over $\mathscr{R}$ and $\mathscr{P}$* correctly enforces *$\mathscr{A}$ iff:*

$$\forall r \in \mathscr{R}, \forall p \in \mathscr{P} : r \in \mathsf{cap}(p) \Leftrightarrow \exists \text{ a path in } G \text{ from } v_p \text{ to } v_r.$$

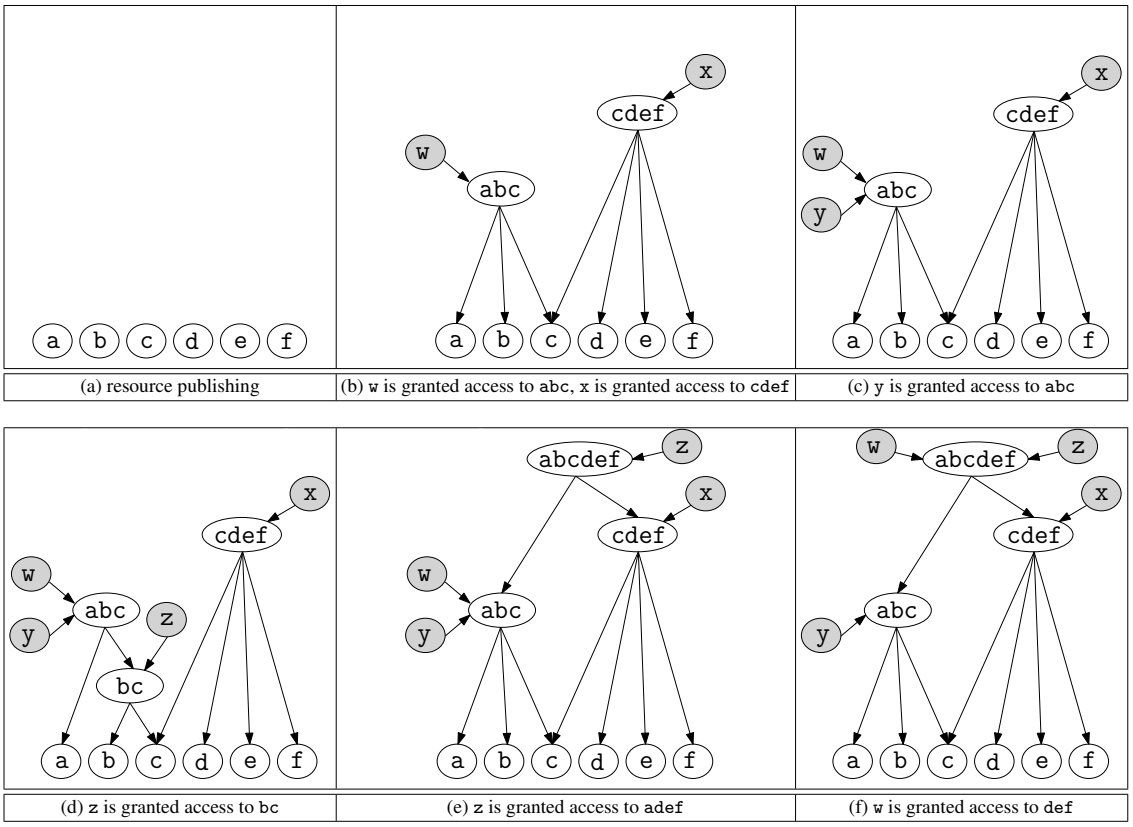
To correctly enforce the authorization policy $\mathscr{A}$, we include in the key derivation structure a vertex for each capability list of the processors in $\mathscr{P}$ and an edge to connect the vertex v_p of each processor to the vertex $v_{\mathsf{cap}(p)}$ representing her capability list $\mathsf{cap}(p)$. If needed to further reduce the number of tokens, we insert additional vertices representing subsets of resources even if not corresponding to any capability list. We then connect vertices representing sets of resources in such a way to ensure that the set R of resources represented by a vertex v_R is *covered* by the vertices directly reachable from v_R through an edge in G (meaning that the union of the sets of resources represented by the vertices directly reachable from v_R is exactly R).

To illustrate, consider the authorization policy in Figure 3.3(a) for the sets of resources and processors of our running example. Figure 3.3(b) illustrates an example of a key derivation structure enforcing the policy. The structure includes one vertex for each resource, one vertex for each processor, and three vertices representing their capability lists. Edges of the structure connect processors to their capability lists, and sets of resources according to the subset containment relationship, in such a way to guarantee coverage (e.g., vertex `abc` is covered by `a`, `b`, and `c`). It is immediate to verify that the key derivation structure in Figure 3.3(b) correctly enforces the authorization policy in Figure 3.3(a), since it enables each processor to reach all and only the resources she is entitled to access.

3.4.2 Resources and access management

The key derivation structure is updated whenever new resources are published and/or new access privileges are granted. The publication of resource r is easily enforced by simply inserting in the structure a vertex for r. The owner generates an encryption key k_r and a label l_r for the vertex, encrypts r with k_r, and publishes the encrypted resource on the market. To illustrate, consider the structures in Figure 3.4. The first structure (Figure 3.4(a)) represents the structure for our running example after the publication of the six resources (since no authorization has been granted yet, no processor vertex is present).

Granting access for a set R of resources to processor p is enforced by procedure **Grant_Access** (Figure 3.5). Note that, in the figure and in the following discussion, the generation of vertices (edges, resp.) implies the generation of their keys and labels (tokens, resp.). The procedure takes as input the requesting processor p, its current capability list $\mathsf{cap}(p)$, the set R of resources she is to be granted access, and the key derivation structure $G(V,E)$. It updates the structure enabling p to derive the keys necessary to decrypt the resources in $\mathsf{cap}(p) \cup R$. The procedure fist checks whether the structure already contains a vertex v_p for p (i.e., if p can already access resources in the market) and, if this is not the case, it creates vertex v_p and the corresponding key and label (lines 1–2). If the vertex $v_{\mathsf{cap}(p)}$ representing the (old) capability list of p does exist, the procedure deletes $(v_p, v_{\mathsf{cap}(p)})$ (lines 3–4). It then checks whether the removal of $v_{\mathsf{cap}(p)}$ could reduce the number of edges [DFJ+10] and, if this is the case, it removes $v_{\mathsf{cap}(p)}$ connecting all its parents to all its children (lines 5–13). The procedure then updates the capability list $\mathsf{cap}(p)$ by including the resources in R (line 14). If the key derivation structure already includes vertex $v_{\mathsf{cap}(p)}$ representing the new capability list of p, then v_p is simply connected to $v_{\mathsf{cap}(p)}$, and the procedure terminates



(a) resource publishing | (b) w is granted access to abc, x is granted access to cdef | (c) y is granted access to abc

(d) z is granted access to bc | (e) z is granted access to adef | (f) w is granted access to def

Figure 3.4: Evolution of a key derivation structure

(line 15). Otherwise, $v_{\mathsf{cap}(p)}$ first needs to be created, and only at this point an edge is created to enable the derivation of $v_{\mathsf{cap}(p)}$ from v_p (lines 16–19). To guarantee the correctness of the key derivation structure, all the resources in $\mathsf{cap}(p)$ should be reachable from $v_{\mathsf{cap}(p)}$ (Definition 3.4.2). Hence, the procedure identifies the set *Desc* of vertices representing subsets of resources in $\mathsf{cap}(p)$, and selects a subset *Cover* of vertices in *Desc* forming a set cover for $\mathsf{cap}(p)$ (lines 20–21). Vertex $v_{\mathsf{cap}(p)}$ is connected to the vertices in *Cover* (line 22). The procedure finally checks if it is possible to further reduce the number of edges [DFJ+10] thanks to the presence of $v_{\mathsf{cap}(p)}$. To this aim, the procedure identifies the set *Par* of the vertices representing supersets of $\mathsf{cap}(p)$, and the set of *DescCover* of the vertices reachable from a vertex in *Cover* (lines 23–24). Indeed, if a vertex v_{par} in *Par* is directly connected to more than one vertex in *Cover* and/or in *DescCover*, the insertion of $v_{\mathsf{cap}(p)}$ as an intermediate vertex and removal of edges from v_{par} to the vertices in *Cover* and/or in *DescCover* (lines 25–31) reduces the number of edges.

Figure 3.4 illustrates the evolution of the first structure (Figure 3.4(a)) to enforce the sequence of requests illustrated at the bottom of the structures. The first two requests insert vertices `abc` and `cdef` for `w` and `x` respectively (Figure 3.4(b)). The third request does not insert vertices for resources since $\mathsf{cap}(\mathtt{y})=\mathtt{abc}$ already belongs to the structure (Figure 3.4(c)). The fourth request inserts `bc` for `z`. Note that connecting `abc` to `bc` saves an edge (Figure 3.4(d)). The fifth request inserts a vertex for the entire set $\mathscr{R}$, for which `z` is authorized. Vertex `bc` then becomes redundant and is removed (Figure 3.4(e)). The last request authorizes `w` for all the resources. Since $\mathsf{cap}(\mathtt{w})$ belongs to the structure, no vertex is inserted (Figure 3.4(f)).

GRANT_ACCESS(p,$\mathsf{cap}(p)$,R,$G(V,E)$)

1: **if** $v_p \notin V$ **then**
2: generate v_p; $V := V \cup \{v_p\}$
3: **if** $v_{\mathsf{cap}(p)} \in V$ **then**
4: $E := E \setminus \{(v_p, v_{\mathsf{cap}(p)})\}$
5: **if** $\nexists p' \neq p$ s.t. $\mathsf{cap}(p) = \mathsf{cap}(p')$ **then**
6: let *par* be the number of incoming edges of $v_{\mathsf{cap}(p)}$
7: let *desc* be the number of outgoing edges of $v_{\mathsf{cap}(p)}$
8: **if** $(par \times desc) < (par + desc)$ **then**
9: **for each** $v_{par} \in V \ : \ (v_{par}, v_{old}) \in E$ **do**
10: $E := E \setminus \{(v_{par}, v_{old})\}$
11: **for each** $v_{desc} \in V \ : \ (v_{old}, v_{desc}) \in E$ **do**
12: $E := E \setminus \{(v_{old}, v_{desc})\} \cup \{(v_{par}, v_{desc})\}$
13: $V := V \setminus \{v_{\mathsf{cap}(p)}\}$
14: $\mathsf{cap}(p) := \mathsf{cap}(p) \cup R$
15: **if** $v_{\mathsf{cap}(p)} \in V$ **then** $E := E \cup \{(v_p, v_{\mathsf{cap}(p)})\}$
16: **else**
17: generate $v_{\mathsf{cap}(p)}$
18: $V := V \cup \{v_{\mathsf{cap}(p)}\}$
19: $E := E \cup \{(v_p, v_{\mathsf{cap}(p)})\}$
20: let $Desc \subseteq V$ be the set of vertices over a set of resources $\subset \mathsf{cap}(p)$
21: let *Cover* be a subset of *Desc* whose resources form a set cover for $\mathsf{cap}(p)$
22: **for each** $v_{cover} \in Cover$ **do** $E := E \cup \{(v_{\mathsf{cap}(p)}, v_{cover})\}$
23: let $Par \subseteq V$ be the set of vertices over a set of resources $\supset \mathsf{cap}(p)$
24: let *DescCover* be the set of vertices reachable from vertices in *Cover*
25: **for each** $v_{par} \in Par$ **do**
26: $ToRemove := \emptyset$
27: **for each** $v \in DescCover \cup Cover$ **do**
28: **if** $(v_{par}, v) \in E$ **then** $ToRemove := ToRemove \cup \{(v_{par}, v)\}$
29: **if** $|ToRemove| \geq 2$ **then**
30: $E := E \cup \{(v_{par}, v_{\mathsf{cap}(p)})\}$
31: **for each** $(v_{par}, v_z) \in ToRemove$ **do** $E := E \setminus \{(v_{par}, v_z)\}$

Figure 3.5: Procedure managing access grants

3.5 Counteracting misbehaviors

When granting an authorization implies an incentive to be paid to the data owner, data owners and processors should trust each other, like in any situation where a vendor sells a product or a service (access to resources, in our case) to a buyer. If access to a resource is granted *before* payment, the owner needs to trust the processor to finalize the payment. If access is granted *after* payment, the processor needs to trust the owner to grant access to the resource(s) for which she paid the incentive. Requiring such level of trust is unrealistic in our scenario, and processors and owners might misbehave to get advantages. We then need a solution enabling them to conclude a contract without fully trusting each other. In this section, we first illustrate possible misbehaviors that a

malicious party could adopt to gain advantages over the counterpart. We then present our solution to mitigate these risks.

3.5.1 Malicious behaviors

In principle, both the data owner and the processors might get advantages in behaving maliciously. The data owner might not grant access to her resources after having received a payment, with economic/privacy advantages. The processor might instead not pay after having received access to a resource, with economic/knowledge advantages. Also, a malicious party can blame a misbehavior on the other (honest) party (e.g., a malicious owner could claim a payment has not been executed and require a new payment). The main misbehaviors can be classified as follows:

- NO_ACCESS: a malicious data owner does not grant access to a processor p for (at least) a resource for which p paid the agreed amount;
- NO_PAYMENT: a malicious processor does not pay to the data owner o the agreed amount for (at least) a resource for which o provided access;
- NO_ACCESS*: a malicious processor claims that, for (at least) a resource for which she paid the agreed amount, access has not been provided (while it has);
- NO_PAYMENT*: a malicious owner claims that, for (at least) a resource for which she granted access to a processor, payment has not been finalized (while it has).

Misbehaviors related to payments (i.e., NO_PAYMENT and NO_PAYMENT*) can be easily prevented by adopting *blockchain* and *smart contracts*, granting access to a resource *upon* the reception of a money transfer from a processor. A straightforward solution could consist of directly trading the encryption keys with a smart contract which, upon receiving a payment from a processor p for a set R of resources, triggers the algorithm illustrated in the previous section to automatically generate keys, labels, and tokens enabling p to access all resources in R. Unfortunately, this solution is not viable due to the public nature of the content of smart contracts. Updating the token catalog requires in fact knowledge of the keys used in the system (including those assigned to processors and those used to encrypt resources), and hence any subject observing the blockchain would be able to decrypt the resources. We now illustrate our solution to this problem.

3.5.2 Counteracting approach

We propose an *interaction protocol*, to regulate the interplay between processors and data owners, and an *audit process*, to identify misbehaving parties. The interaction protocol prevents NO_PAYMENT and NO_PAYMENT* misbehaviors, ensuring that if an incentive must be paid to the owner for accessing some of her resources, then no access can be granted without such payment. The audit process detects NO_ACCESS and NO_ACCESS* misbehaviors, exposing malicious owners (processors, resp.) that do no grant access despite having received the incentive (maliciously claim an access for which they paid the incentive has not been granted, resp.). The combined adoption of these two approaches incentivizes all parties to behave correctly.

Interaction protocol. The interaction protocol relies on smart contracts to regulate how a processor p and a data owner o should operate to safely finalize the payment of the incentive following an access grant to a set R of resources. Its adoption guarantees that the data owner receives the

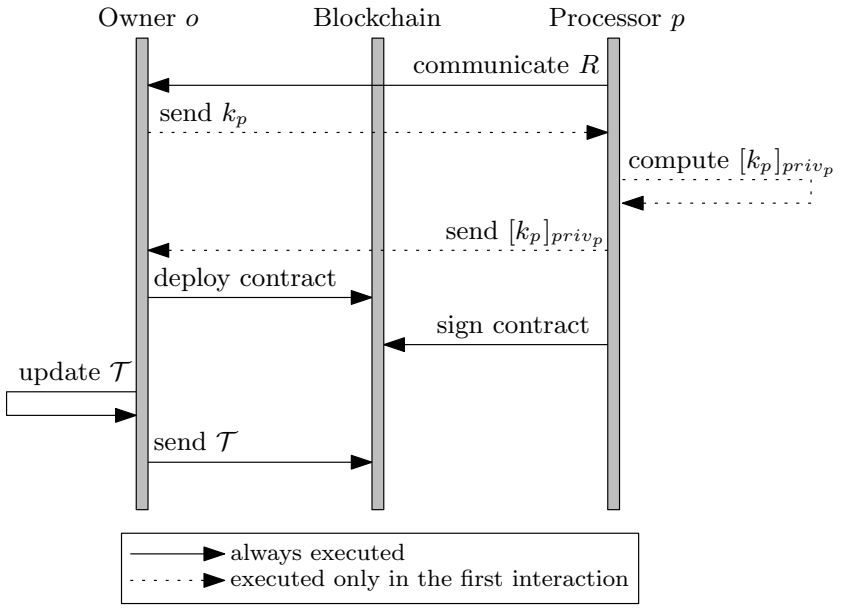


Figure 3.6: Interaction protocol

incentive and the processor receives a public commitment by the data owner to grant access to R. Since encryption keys cannot be directly managed through smart contracts, we leverage smart contracts and blockchains only to enforce the payment of the incentive, and to securely log the willingness of o to grant p access to the requested resources *after* the payment is received. An executed smart contract then represents an incontrovertible proof that: *i)* the payment has been performed; and *ii)* the data owner is aware of her obligation to give p access to the resources. We then complement smart contracts with a solution (i.e., the audit process) that enables a designated trusted subject (i.e., an auditor) to check, upon request (e.g., when one of the two parties detects or suspects a misbehavior), whether the accesses dictated by the contract are actually provided (i.e., whether the owner did what she committed to). To enable such control, within the interaction protocol we: *i)* store on-chain the catalog $\mathcal{T}$ and the capability lists of the processors (so that they can be queried in the audit); and *ii)* require the encryption key k_p provided by o to p to be signed by p (so that it can be proved to be authentic in the audit). We then assume each processor to have a private ($priv_p$), public (pub_p) key pair.

The interaction between a processor p, willing to purchase a set R of resources, and the owner o of the resources operates according to the protocol in Figure 3.6. Note that we consider the case in which o is willing to grant p access for R: if this is not the case and the owner does not wish to grant access, she simply does not participate in the protocol, and the interaction terminates. The protocol operates as follows:

1. p contacts o (off-chain) communicating the set R of resources for which she requests access and for which she is willing to pay the incentive to o;

2. if p and o have not yet interacted, o generates a key k_p for p and sends it to p;

3. upon receiving k_p, p signs it with her secret key $priv_p$, and sends the signed key (denoted $[k_p]_{priv_p}$) to o;

4. o prepares a smart contract, dictating that "upon receiving `incentive` from p for R,

$\mathsf{cap}(p) := \mathsf{cap}(p) \cup R$, and the token catalog $\mathscr{T}$ is updated in such a way to allow p to derive the keys for the resources in $\mathsf{cap}(p)$";

5. o deploys the contract on the blockchain;
6. p accesses the contract and signs it, automatically triggering the payment for `incentive`;
7. o updates the key derivation structure as required by the executed contract (see Section 3.4.2);
8. o updates $\mathscr{T}$ on the blockchain.

Since $\mathscr{T}$ is stored on-chain, at the end of the interaction protocol p can query it for obtaining the information necessary to derive the keys for the resources in R. Also, since the key derivation structure is updated by the owner on her premises, keys are kept safe. Note that, to enable the audit process (as clarified in the remainder of this section), tokens operate on signed keys $[k_p]_{priv_p}$ (in contrast to k_p).

If an interested processor and a data owner interact through this protocol, both NO_PAYMENT and NO_PAYMENT* misbehaviors are prevented. In fact, the key derivation structure is updated locally by the owner granting the processor access to resources only *after* the payment has been received. Since the blockchain is public, every user can verify whether the payment has been performed.

Audit process. Since accesses are directly granted by the data owner and keys and resources are not exchanged on-chain, NO_ACCESS and NO_ACCESS* misbehaviors cannot be prevented. We then propose an audit process for detecting and exposing them (hence negatively impacting on the reputation of the misbehaving party). To this end, our audit process allows a designated trusted auditor, arbitrarily agreed between the owner and the processor (and possibly identified in the smart contract, so to have a proof that both parties agree on it), to check whether the processor does have access to all the resources for which she paid the incentive. The audit process can be invoked either by a processor claiming and wishing to expose a NO_ACCESS misbehavior, or by an owner claiming and wishing to expose a NO_ACCESS* misbehavior.

Given the identity of the processor p and of the owner o involved in the audit process, the auditor checks whether the current token catalog $\mathscr{T}$ enables p to derive the keys for the resources in $\mathsf{cap}(p)$ to discriminate between NO_ACCESS and NO_ACCESS*. Figure 3.7 illustrates the audit process. The auditor first needs to query the public ledger maintained on-chain to obtain: the capability list $\mathsf{cap}(p)$ of the processor, the token for deriving key $k_{\mathsf{cap}(p)}$, and label $l_{\mathsf{cap}(p)}$. If the token (or the label) does not exist, the auditor signals a NO_ACCESS misbehavior. In fact, the derivation structure cannot allow p to derive the keys for the resources in $\mathsf{cap}(p)$ (i.e., the owner ignored the incentive). Otherwise, the auditor retrieves $[k_p]_{priv_p}$ from the owner and, using the catalog, derives all the encryption keys reachable from $[k_p]_{priv_p}$. To verify that $[k_p]_{priv_p}$ is the key that p and o exchanged in the interaction protocol (Figure 3.6), the auditor checks its signature. If signature verification fails, either o has defined/updated the derivation structure starting from the wrong key (and hence p cannot access the resources in her capability list), or it is not participating honestly in the audit process (i.e., she returned a different key from the one agreed with p). In either case, the auditor signals a NO_ACCESS misbehavior, exposing a misbehavior of the owner. On the contrary, if signature verification succeeds, the auditor derives $k_{\mathsf{cap}(p)}$ and the keys of the resources in $\mathsf{cap}(p)$. The auditor can then try to decrypt all resources in $\mathsf{cap}(p)$ using these keys. If decryption fails (because for at least one resource the related key is not derivable or incorrect), the auditor again signals a NO_ACCESS misbehavior. Otherwise, if decryption succeeds, the auditor returns

AUDIT(p,o)
1: **retrieve** $\mathsf{cap}(p)$
2: **retrieve** $t_{p,\mathsf{cap}(p)}$ and $l_{\mathsf{cap}(p)}$ from $\mathscr{T}$
3: **if** $t_{p,\mathsf{cap}(p)}$ = NULL **or** $l_{\mathsf{cap}(p)}$=NULL **then**
4: **return** 'NO_ACCESS misbehavior'
5: **retrieve** $[k_p]_{priv_p}$ from o
6: **if** signature verification of $[k_p]_{priv_p}$ fails **then**
7: **return** 'NO_ACCESS misbehavior / o not collaborating'
8: **compute** $k_{\mathsf{cap}(p)} = h([k_p]_{priv_p}, l_{\mathsf{cap}(p)}) \oplus t_{p,\mathsf{cap}(p)}$
9: **derive** all the resource keys k_r derivable from $k_{\mathsf{cap}(p)}$
10: **for each** $r \in \mathsf{cap}(p)$ **do**
11: $r :=$ **decrypt**$(E(r), k_r)$
12: **if** decryption fails **then**
13: **return** 'NO_ACCESS misbehavior'
14: **return** 'NO_ACCESS* misbehavior'

Figure 3.7: Pseudocode of the audit process

a NO_ACCESS* misbehavior, since the owner respected her obligation and hence the processor is dishonestly accusing the owner of misbehavior. Note that the audit process could expose the plaintext content of resources to the auditor, when the processor maliciously accuses the data owner of misbehavior. However, a malicious processor can disclose the purchased resources to any subject (hence including the auditor) independently from the audit process, so the process itself does not introduce additional disclosure risks. Also, thanks to our audit process, the misbehavior of the processor is revealed. Therefore, we expect processors not to dishonestly blame a misbehavior on an honest owner, as this would decrease their reputation.

The reliability of the results of the audit process clearly depends on the correctness and freshness of the data over which controls operate (i.e., tokens, labels, capability lists, processors' signed keys). The correctness and freshness of tokens, labels, and capability lists is guaranteed by the fact that they are stored on-chain, as dictated by the smart contract, and hence in a safe and immutable ledger that the auditor can query. The correctness and freshness of $[k_p]_{priv_p}$ is guaranteed by the digital signatures, since o cannot reproduce (nor p repudiate) a signature with $priv_p$.

The availability of the audit process clearly incentivizes both the data owner and the processors to behave correctly. Indeed, the audit process reveals misbehaviors and publicly exposes the identity of the malicious subject. This can have serious consequences on her reputation, with clear damages in the data market where (in a similar way to, for instance, e-commerce platforms) lower reputation can be expected to cause lower willingness of other parties to engage in interactions. We then expect the availability of our audit process to prevent misbehaviors.

3.6 Discussion

The combined adoption of selective encryption and of an approach based on blockchain, smart contracts, and an audit process for regulating the interactions between data owners and processors can set a first step towards the enforcement of transparent processing. Transparent processing of

data implies, among other aspects, to log all events related to data processing and sharing, and to enable the control that the processing itself is performed according to the policy set by the data owner [BKPW17]. We fulfill these requirements by ensuring that: *i)* each resource is shared only with processors authorized by the owner; and *ii)* sharing is securely logged on-chain, producing a verifiable trail of sharing history. This ensures that the data owner knows and can prove, at any time, who is able to access her resources, and that each processor is able to prove that all accesses to resources were authorized. Also, since we store both the authorization policy and the token catalog on-chain, their updates leave a permanent trace. Hence, not only is it possible to verify the enforcement of the most up-to-date policy, but also past versions of the token catalog can be checked for verifying the correct enforcement of a former one. Also, since resources are not stored on-chain, they can always be deleted (also in accordance to the EU GDPR).

We close this section with a note on the generality of our solution. Our proposal does not rely on specific technologies or architectures, and hence can be easily tailored and deployed in different application scenarios. For instance, resource protection through selective encryption (Section 3.4) can operate with arbitrary (sufficiently strong) encryption schemes. Similarly, we do not restrict our smart contracts to operate on a specific blockchain (e.g., Ethereum) or programming language for coding the smart contract. Of course, the security of the overall system depends on the correctness of the developed code, like in any interaction governed by a smart contract.

3.7 Summary

This chapter focused on the problem of enforcing controlled sharing through data wrapping in the data market scenario. The approach illustrated in this chapter leverages selective owner-side encryption to protect the confidentiality of the data outsourced to the market, and key derivation for allowing authorized parties to compute the keys necessary to unwrap and access plaintext resources. We also considered the management of incentives to data owners, and proposed a solution based on blockchain/smart contracts and an audit process for counteracting possible misbehaviors. MOSAICrOWN is now studying the definition of novel strategies for managing keys and key derivation.

4. Conclusions

This document presented the results of research work performed in MOSAICrOWN Work Package 4 devoted to the design and development of advanced encryption-based solutions for wrapping data ingested, stored, processed, or shared in the digital data market. The goal is to provide efficient solutions enjoying strong protection guarantees while preserving access and processing functionality. The techniques presented in this deliverable address the challenges entailed by such a goal tackling different problems related to ensuring data protection and functionality in the different phases of the information life cycle. They enable data owners, curators, and processors to properly enjoy functionality over the data while ensuring proper data protection against non authorized users or misbehaving parties. The modular techniques presented in this deliverable represent building blocks towards the realization of a digital data market empowering data owners, in compliance with data protection regulations and policies.

The techniques presented in Chapter 1 enable owners to enjoy availability of the data market for processing data in collaborative computations, possibly leveraging economical solutions and involving external parties in the computation, while ensuring data are not leaked (directly or indirectly) to non authorized parties. Such protection is realized by the controlled ingestion of encryption/decryption operations in the execution of query plans, as needed to cover the data against unauthorized parties to which parts of the computation are assigned.

The techniques presented in Chapter 2 offer an advanced All-Or-Nothing-Transform (AONT) encryption providing for strong protection guarantees even in cases where encryption key may be leaked. They also enable the efficient realization of revocation for large resources without requiring their complete re-encryption. The chapter also presents our approach to enrich the management of AONT-encrypted resources leveraging the availability of decentralized data storage. Our solution provides for controlled slicing and allocation of resources, enabling owners to set the parameters regulating them as it best suits their needs.

The techniques presented in Chapter 3 enable data owners to control encryption when ingesting data in the data market, wrapping them in a protection layer that can be accessible only to authorized consumers. The consideration of blockchain and smart contracts provides owners with full control over their resources, permitting access to them against payment, towards the realization of a digital data market supporting economic incentives.

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.

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

[AJJP16] M. Albanese, S. Jajodia, R. Jhawar, and V. Piuri. Dependable and resilient cloud computing. In *Proc. of IEEE SOSE*, Oxford, UK, March 2016.

[ATL15] A. Aldribi, I. Traore, and G. Letourneau. Cloud slicing a new architecture for cloud security monitoring. In *Proc. of IEEE PACRIM*, Victoria, Canada, August 2015.

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

[BDF+16] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati. Mix&slice: Efficient access revocation in the cloud. In *Proc. of ACM CCS*, Vienna, Austria, October 2016.

[BDF+19a] E. Bacis, S. De Capitani di Vimercati, D. Facchinetti, S. Foresti, G. Livraga, S. Paraboschi, M. Rosa, and P. Samarati. Multi-provider secure processing of sensors data. In *Proc. of PerCom*, Kyoto, Japan, March 2019.

[BDF+19b] 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 IEEE GLOBECOM*, Waikoloa, Hawaii, USA, December 2019.

[BDF+20] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati. Securing resources in decentralized cloud storage. *IEEE TIFS*, 15(1):286–298, December 2020.

[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), 2017.

[BJO09a] K.D. Bowers, A. Juels, and A. Oprea. HAIL: A high-availability and integrity layer for cloud storage. In *Proc. of ACM CCS*, Chicago, IL, USA, November 2009.

[BJO09b] K.D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: Theory and implementation. In *Proc. of ACM CCSW*, Chicago, IL, USA, November 2009.

[BKPW17] P. Bonatti, S. Kirrane, A. Polleres, and R. Wenning. Transparent personal data processing: The road ahead. In *Proc. of SAFECOMP*, Trento, Italy, September 2017.

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

[BMMC14] M. Baldi, N. Maturo, E. Montali, and F. Chiaraluce. AONT-LT: A data protection scheme for cloud and cooperative storage systems. In *Proc. of HPCS*, Bologna, Italy, July 2014.

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

[CSGM06] P. Cataldi, M. P. Shatarski, M. Grangetto, and E. Magli. Implementation and performance evaluation of LT and Raptor codes for multimedia applications. In *Proc. of IEEE MSP*, Pasadena, CA, USA, December 2006.

[CZK+19] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *Proc. of IEEE EuroS&P*, Stockholm, Sweden, June 2019.

[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.

[DEF18] S. Dziembowski, L. Eckey, and S. Faust. FairSwap: How to fairly exchange digital goods. In *Proc. of ACM CCS*, Toronto, Canada, January 2018.

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

[DFLS16] S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati. Practical techniques building on encryption for protecting and managing data in the cloud. In P. Ryan, D. Naccache, and J.-J. Quisquater, editors, *The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*. Springer, 2016.

[DFLS19] S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati. Empowering owners with control in digital data markets. In *Proc. of IEEE CLOUD*, Milan, Italy, July 2019.

[DMR17] D. Di Francesco Maesa, P. Mori, and L. Ricci. Blockchain based access control. In *Proc. of DAIS*, Neuchâtel, Switzerland, June 2017.

[DPNH20] S. Delgado-Segura, C. Pérez-Sola, G. Navarro-Arribas, and J. Herrera-Joancomartí. A fair protocol for data trading based on bitcoin transactions. *FGCS*, 107:832–840, 2020.

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

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

[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, June 2002.

[Irv10] D. Irvine. Maidsafe distributed file system. Technical report, MaidSafe, 2010.

[KAS+18] E Kokoris-Kogias, EC Alp, SD Siby, N Gailly, L Gasser, P Jovanovic, E Syta, and B Ford. CALYPSO: Auditable sharing of private data over blockchains. Technical report, Cryptology ePrint Archive, Report 2018/209, 2018.

[KB16] 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, 2000.

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

[NAL14] D. Nuñez, I. Agudo, and J. Lopez. Delegated access for hadoop clusters in the cloud. In *Proc. of IEEE CloudCom*, Singapore, December 2014.

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

[PGK88] D.A. Patterson, G. Gibson, and R.H. Katz. A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD Records*, 17(3):109–116, June 1988.

[PHI14] G. Paul, F. Hutchison, and J. Irvine. Security of the maidsafe vault network. In *Wireless World Research Forum Meeting 32*, Marrakesh, Morocco, May 2014.

[PRZB11] 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.

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

[Ron97] L. R. Ronald. All-or-nothing encryption and the package transform. In *Proc. of FSE*, Haifa, Israel, January 1997.

[RP11] J. K. Resch and J. S. Plank. AONT-RS: Blending security and performance in dispersed storage systems. In *Proc of FAST*, San Jose, CA, USA, February 2011.

[RS60] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, June 1960.

[SBHD17] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy. Towards blockchain-based auditable storage and sharing of IoT data. In *Proc. of CCSW*, Dallas, TX, USA, November 2017.

[Sho11] Amin Shokrollahi. Raptor codes. *IEEE/ACM TON*, 6(3-4):213–322, May 2011.

[SKS+19] G. Salvaneschi, M. Köhler, D. Sokolowski, P. Haller, S. Erdweg, and M. Mezini. Language-integrated privacy-aware distributed queries. *PACMPL*, 3(OOPSLA), October 2019.

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

[TPPG13] M. Theoharidou, N. Papanikolaou, S. Pearson, and D. Gritzalis. Privacy risk, security, accountability in the cloud. In *Proc. of IEEECloudCom*, Bristol, UK, December 2013.

[VC14] D. Vorick and L. Champine. Sia: Simple decentralized storage. Technical report, Nebulous Inc., 2014.

[WBB+16] S. Wilkinson, T. Boshevski, J. Brandoff, J. Prestwich, G. Hall, P. Gerbes, P. Hutchins, C. Pollard, and V. Buterin. Storj: a peer-to-peer cloud storage network (v2.0). Technical report, Storj Labs Inc., 2016.

[WM01] M. Waldman and D. Mazieres. Tangler: a censorship-resistant publishing system based on document entanglements. In *Proc. of ACM CCS*, Philadelphia, PA, USA, November 2001.

[ZNP15] G. Zyskind, O. Nathan, and A. Pentland. Decentralizing privacy: Using blockchain to protect personal data. In *Proc. of IEEE SPW*, San Jose, CA, USA, May 2015.

[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, 2015.