This version of the contribution has been accepted for publication, after peer review (when applicable) but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: https://doi.org/10.1007/978-3-031-36840-0_17. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms

Selective Encryption for Owners' Control in Digital Data Markets

Sara Foresti[0000−0002−1658−6734] and Giovanni Livraga[0000−0003−2661−8573]

Università degli Studi di Milano, Italy
{sara.foresti,giovanni.livraga}@unimi.it

Abstract. A recent solution for permitting data owners to make their data selectively available on digital market platforms combines the adoption of selective owner-side encryption and smart contracts deployed on blockchains. Selective encryption, coupled with key derivation techniques to reduce key management burden, guarantees that only subjects who are entitled to access a resource can read its content. The adoption of smart contracts deployed on a blockchain permits to regulate the interplay among the interacting parties, the possible economic incentives to be paid to the owners as expected in markets, and the exchange of the information necessary for resource decryption (i.e., updates to the key derivation structure) upon payment. In this paper, we investigate different approaches for managing and updating the key derivation structure to enable selective data sharing on digital data markets, while limiting access times to resources and the cost of operations on the blockchain.

Keywords: Digital data market · Selective encryption · Key derivation.

1 Introduction

An ever-increasing deal of interest focuses nowadays towards the development and adoption of spaces and platforms where data can be easily shared and/or traded among interested subjects. Such spaces, typically called digital data markets, represent virtual places where data owners (acting as data producers) offer datasets, and data consumers can access (parts of) them. The creation of these platforms, enabling data sharing among different subjects, can have a positive impact on the creation of knowledge based on the analysis of heterogeneous data, with clear societal benefits.

One of the main concerns that can hinder the adoption of digital data markets is the (perceived) lack of control owners can suffer resorting to these platforms. Some concerns naturally arise in any scenario in which a data owner wishes to delegate storage and management of data to an external third party that can be considered honest-but-curious (i.e., trusted for correctly managing the data, but not trusted for accessing their content) or, more in general, not fully trusted (e.g., it can be considered lazy). In addition, the scenario of digital data markets complicates the picture with peculiarities that require careful consideration, such as the possibility of providing owners with payments when consumers access their

data as a reward for contributing their data. Clearly, ensuring proper protection to the data managed and shared in digital data markets, and ensuring owners remain in control over they data obtaining rewards when it is the case, are key requirements for enabling a wide adoption of such markets.

To permit the adoption of digital data markets to trade data while ensuring data are protected an under the control of their owner, without the need of completely trusting the market provider, a recent proposal combines selective owner-side encryption and blockchain [6]. With selective owner-side encryption, data are published on the market only after being encrypted by their owner. Encryption is performed using different encryption keys, so that different data items are encrypted with different keys. Encryption keys are distributed to consumers according to the restrictions set by the owners, including the fact of having received a reward. In this way, encrypted data can be possibly stored directly on the premises of the data market, if available, or more generally of economically-convenient cloud platforms with the guarantee that unauthorized subjects (including the market/cloud provider itself) cannot access their plaintext content. When an interested consumer requests access to a certain dataset, the owner/consumer interaction and the possible economic transaction for the incentives are managed via smart contracts deployed on a blockchain. In this way the request, the payment, and the willingness of the owner to grant access to such data to the consumer remains logged in the immutable ledger of the blockchain, ensuring therefore transparency and accountability.

The combined adoption of blockchain and selective owner-side encryption should however be carefully regulated. To ensure a reasonable key management burden for consumers, selective owner-side encryption is typically complemented by key derivation techniques, which define key derivation structures that represent sets of tokens enabling the derivation (i.e., the computation) of one encryption key starting from another one. The catalog of the tokens is publicly stored on the blockchain so that key derivation can be executed in accordance with the restrictions imposed by the owners. Every time an owner grants access to a new data item, or set thereof, the key derivation structure (and hence the entailed token catalog) needs to be restructured to reflect and enforce the new granted access. Different optimization strategies can be pursued when updating a key derivation structure. We investigate this issue and propose two different strategies: while the first aims at maintaining a slim token catalog, trying to minimize the overall number of tokens inserted in the catalog to ensure fast retrieval of tokens, the second strategy we investigate aims at reducing the modifications to the catalog, trying to reduce the costs entailed by updating the catalog on-chain. A preliminary version of this work appeared in [8], which we here extend with more complete and revised algorithms, and enhanced discussions on the possible strategies that can be adopted along with their pros and cons. The remainder of this paper is then organized as follows. Section 2 illustrates our reference scenario and introduces the general problem of maintaining owners in control in digital data markets. Section 3 provides some preliminaries on the building blocks that our solution adopts. Section 4 discusses how access restrictions specified by the

owner can be self-enforced by the data stored on the market through selective encryption and key derivation. Section 5 illustrates how to selectively encrypt data for being securely stored and traded on data markets. Section 6 illustrates our approach for selectively granting access to data by updating the key derivation structure and token catalog according to our optimization strategies. Finally, Section 7 illustrates related work, and Section 8 concludes the paper.

2 Scenario and Problem Statement

We are concerned with the problem of permitting a secure adoption of digital data markets permitting owners to publish data, and interested consumers to access (some of) them provided that an adequate reward is paid, by the consumers, to the owner. The general reference scenario is then characterized by two sets of subjects: a set $\mathcal{O}$ of data owners on one side, and a set $\mathcal{C}$ of consumers on the other side, who leverage the availability of the a data market to sell and buy data, which for generality we model as a set $\mathcal{R}$ of resources. Formally, our problem can be formulated as follows.

Problem 1. Let $\mathcal{R}$ be a set of resources published on a data market platform, and $\mathcal{C}$ be a set of consumers. $\forall r \in \mathcal{R}, \forall c \in \mathcal{C} : c$ can access r iff c is authorized by o and paid $\texttt{cost}(r)$ to o, with o the owner of r and $\texttt{cost}(r)$ the reward for r.

This problem entails a number of challenges and issues that need to be carefully addressed. A first issue concerns the definition of the rewards to be paid for accessing resources. As it is to be expected in real-world scenarios, we assume such rewards to be defined by the owners of the resources, such that each owner sets the rewards that need to be paid for accessing her resources. For simplicity but without loss of generality, given a resource r published by an owner o, we assume the reward $\texttt{cost}(r)$ to be paid for accessing r to be fixed and equal to each consumer, while we note that our solution is general and is not impacted by different definitions of rewards.

A second issue that characterizes our problem concerns the processing of the payment of the rewards to the owners. In particular, considering the digital nature of data markets and the fact that owners, consumers, and market provider may not fully trust each other, a key requirement demands payments must be correctly executed: a consumer cannot claim to have paid for a resource while she has not and, conversely, an owner cannot claim a payment by a consumer has not been received while it has. A recent solution [6] has put forward the idea of leveraging blockchain and smart contracts to execute payments and have a verifiable log of the accesses requested and granted. We build on this strategy and leverage blockchain and smart contracts to guarantee that *i)* after a consumer c purchases access to a resource r, by paying $\texttt{cost}(r)$ to r's owner o, then o cannot claim that payment has not been received and refuse to grant access to c; and *ii)* after an owner o has granted to a consumer c access to a resource r, then c cannot claim that access has not been granted (and ask to be refunded $\texttt{cost}(r)$).

A third issue that characterizes our problem, and which represents the main focus of this paper, concerns ensuring adequate data protection, meaning that only authorized consumers who have paid adequate rewards to the owners can access resources. A complicating factor in this regard is that the market provider may not be fully trusted to enforce access restrictions (i.e., to prevent unauthorized accesses to resources that have not been paid). The provider may in fact be lazy (and not enforce access control, partially or entirely) and/or not completely trusted to act as a reference monitor. The solution in [6] protects resources with owner-side encryption, so that resources are outsourced to the market in an encrypted form that makes them unintelligible to any subject who does not possess the correct encryption keys, and in providing authorized consumers with the correct encryption keys. In our work we leverage selective owner-side encryption (for supporting fine-grained access restrictions) and key derivation (to ease key management to consumers) to ensure that only authorized consumers can access a resource, without relying on the active involvement of the market provider.

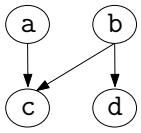
Example 1. We refer our discussion to the data generated by a smart fitness device that measures different parameters (e.g., heart rate, steps and movements, sleep, oxygen saturation levels). Such measurements are collected throughout the whole day, and the owner of the device and of the measured data can download them. Since these data can be of interest to a multitude of subjects (e.g., researchers), the owner decides to monetize them and, at regular time intervals, publishes them on a data market platform. In our examples, we consider the release of five sets of measurements $\mathcal{R}=\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e}\}$. We consider four consumers $\mathcal{C}=\{\alpha,\beta,\gamma,\delta\}$, which over time require access to resources and hence pay rewards to the owner.

3 Preliminaries and Sketch of the Approach

The solution studied in this paper combines *selective encryption* (for protecting resources) and *key derivation* (for permitting efficient management), with *blockchain* and *smart contracts* (for managing the payments among possibly distrusting parties). In this section we provide some preliminaries on these building blocks.

3.1 Selective Encryption and Key Derivation

Encrypting the resources to be stored at an external platform is an effective approach for protecting their confidentiality, since the encryption layer (set by the data owner) makes resources unintelligible to subjects who do not know the encryption key, possibly including the provider itself. The enforcement of fine-grained access restrictions can be effectively managed through selective encryption [1, 4] whenever the storage platform is not considered trusted to mediate access requests. Selective encryption consists in wrapping each resource with an encryption layer (set by the owner) using different keys for different resources,



$\mathcal{T}$

From	To	Value
a	c	$k_{\mathtt{c}} \oplus h(k_{\mathtt{a}}, l_{\mathtt{c}})$
b	c	$k_{\mathtt{c}} \oplus h(k_{\mathtt{b}}, l_{\mathtt{c}})$
b	d	$k_{\mathtt{d}} \oplus h(k_{\mathtt{b}}, l_{\mathtt{d}})$

Fig. 1. An example of key derivation structure and token catalog

and in distributing keys to users in such a way that each user can decrypt all and only the resources she is authorized to access. To mitigate the burden of key management, selective encryption is usually coupled with key derivation [1]. Given two keys k_x and k_y, 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 l_y a public label associated with k_y, $\oplus$ 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 DAGs (directed acyclic graphs), where vertices represent encryption keys and their labels (e.g., vertex v_x represents key k_x), and edges represent tokens (i.e., edge (v_x, v_y) represents token $t_{x,y}$ enabling the derivation of k_y from k_x). Tokens are physically stored in a public catalog $\mathcal{T}$. Key derivation allows each user to manage a single key, from which she can compute all the keys for the resources for which she is authorized. In the following, when clear from the context, we will use the terms keys and vertices (tokens and edges, respectively) interchangeably. Figure 1 illustrates an example of a key derivation structure and of the corresponding token catalog, regulating the derivation relationships among four encryption keys $k_{\mathtt{a}}$, $k_{\mathtt{b}}$, $k_{\mathtt{c}}$, and $k_{\mathtt{d}}$. For readability, we denote the label of an encryption key k_x with x, and use the label to denote its corresponding vertex v_x in the structure (e.g., vertex `a` in Figure 1 represents key $k_{\mathtt{a}}$ and its label). For example, the tokens in the structure permit the derivation of $k_{\mathtt{c}}$ from key $k_{\mathtt{a}}$ and $k_{\mathtt{b}}$, and the derivation of $k_{\mathtt{d}}$ from $k_{\mathtt{b}}$ only.

3.2 Blockchain and Smart Contracts

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, and each block b_i contains a number of transaction records along with a cryptographic hash of the previous one b_{i-1}. 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 then continuously agreed upon by the network of peers: everyone can inspect a blockchain, but no single user can control it, or tamper with it, since modifications to the content of a blockchain require mutual agreement. Once a block is committed, nobody can modify it: updates are reflected in a new block containing the modified information. This permits to trust the content and the status of a blockchain, while not trusting the specific underlying peers. Blockchain is probably best known

to be the core component of Bitcoin for allowing secure value transfer between two parties (through their digital wallets) without the need of a trusted central authority, such as a bank. The cryptographic primitives and digital signatures used for value transfer certify that the value is deducted from the payee's wallet, and that the receiver is indeed the party who now holds the value. Since then, blockchain technology has evolved to the aim of allowing transactions over virtually *any* value, material of immaterial. This has been achieved by introducing the concept of smart contracts, a powerful tool for establishing contracts among multiple, possibly distrusting, parties. A smart contract is a software running on top of a blockchain defining a set of rules, on which the interacting parties agree. It can be seen as a set of 'if-then' instructions, defining triggering conditions (the 'if' part) and subsequent actions (the 'then' part). These conditions capture and model in a machine-readable format the clauses of a contract that is to be signed by the parties. The execution of a smart contact 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 [3].

3.3 Sketch of the Approach

We enforce access restrictions to resources, ensuring their owners remain always in control of who can access them, with owner-side encryption, so that resources are stored in encrypted form to the market, hence being protected also from the eyes of the market provider. Key management burden is reduced for consumers through key derivation, so that each consumer has to manage a single key per owner, starting from which she can derive all the keys needed to decrypt and access all authorized resources. To ensure that key derivation reflects the payments of the rewards enforced by consumers, and to counteract possible misbehaviors from the interacting parties, we follow the proposal in [6] and assume reward payments to occur leveraging a blockchain, and storing the token catalog enabling key derivation on-chain. As will be illustrated in the remainder of this paper, the key derivation structure (and hence the token catalog) needs to be updated whenever a consumer requests (and pays the reward for) a new resource, so to maintain correctness of the enabled key derivations. Different strategies may be adopted for enforcing such updates, which may have different costs that can be considered more or less important depending on the pursued objective. A first and natural objective is to keep the token catalog small in size, so to ensure fast retrieval time by consumers when in need of deriving keys. With this strategy, possibly extensive restructuring operations to the key derivation structure and token catalog may be performed to the aim of reducing the overall number of tokens. Considering that writing on a blockchain has a cost, a second objective is to keep the number of modified tokens small, so to ensure a reduction in the costs entailed by the on-chain storage of the token catalog. In the following sections,

Consumer	Capability List
α	abc
β	cde
γ	a
δ	d

Fig. 2. An example of authorizations for Example 1

we illustrate approaches for enforcing selective encryption and key derivation to protect resources in data markets pursuing these two objectives.

4 Authorizations and Key Derivation

We illustrate how to guarantee that the content of each resource published on the data market is visible only to authorized consumers who paid the reward to their owners. For simplicity, but without loss of generality, we refer our discussion to the set of resources published by one owner o, with the note that the same reasonings apply to all data owners operating on the data market. We illustrate how authorizations can be modeled in Section 4.1, and how they can be enforced through the definition of a key derivation structure in Section 4.2.

4.1 Authorizations

In line with the goal of monetizing data, we assume that a resource r can be accessed by any consumer who paid $\texttt{cost}(r)$ to the owner. For this reason, the authorization policy regulating which consumer can access which resources reflects the payments made by the consumers themselves, but we note that our approach can also manage additional access conditions that a specific owner may wish to impose.

We represent the authorization policy granting access to resources to authorized consumers through the *capability lists* of the consumers in $\mathcal{C}$. Given a consumer $c \in \mathcal{C}$, her capability list $\texttt{cap}(c)$ includes all resources in $\mathcal{R}$ that c can access (i.e., in our scenario, for which c paid $\texttt{cost}(r)$ to the owner). Capability lists are by definition dynamic, and reflect the updates to the authorization policy. In particular, whenever a consumer c purchases access to a new resource r, r will be added to $\texttt{cap}(c)$. In principle, an access (say, to resource r) that has been previously granted to c may also be revoked, resulting in r to be removed from $\texttt{cap}(c)$. We however note that such access revocation is not in line with the peculiarities of the data market platforms, also considering the fact that our reference scenario is characterized by consumers purchasing (and not, for example, renting) access to a resources. For this reason, we do not consider access revocation and, as a consequence, capability lists of consumers never lose elements in the course of time. Figure 2 illustrates an example of authorizations for the consumers and resources of our running example (Example 1). For readability, we represent sets omitting commas and curly brackets (e.g., abc stands for

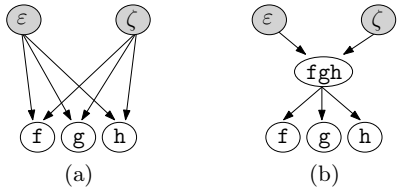


Fig. 3. Two examples of a key derivation structure enforcing $\mathsf{cap}(\varepsilon) = \mathsf{cap}(\zeta) = \mathtt{fgh}$, enabling key derivation through six tokens (a), and five tokens (b)

$\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$). For example, $\mathsf{cap}(\alpha) = \mathtt{abc}$, meaning that consumer α is authorized (and hence has paid the reward to the owner) for resources a, b, and c.

As introduced in Section 3, we protect the confidentiality of the resources and enforce access restrictions through selective owner-side encryption [4]. As highlighted by the name, selective encryption requires the owner to encrypt different resources with different encryption keys before outsourcing them to the market (or, more in general, to an external storage/management platform). In this way, an authorization policy can be easily enforced by providing each consumer c with only the encryption keys necessary to decrypt the resources c is authorized to access.

Fine-grained access restrictions with selective owner-side encryption can be practically enforced in different ways, depending on how encryption keys are managed and distributed to consumers. A naive approach could consist in encrypting each resource $r \in \mathcal{R}$ with a different key k_r, and in distributing to each consumer c the keys used to encrypt all resources in her capability list $\mathsf{cap}(c)$. While this approach can correctly enforce an authorization policy, it would place upon consumers the burden of storing and managing a number of keys linear in the number of resources they can access (i.e., each consumer c would need to manage one key k_r for each resource $r \in \mathsf{cap}(c)$). Key derivation (Section 3.1) can be effectively employed for reducing such key management burden: the possibility of deriving (computing) an unlimited number of encryption keys starting from the knowledge of a single encryption key allows each consumer c to agree with the data owner a single key k_c, and the owner can define and publish a set of $|\mathsf{cap}(c)|$ tokens, each one allowing the computation of the encryption key used for a resource in $\mathsf{cap}(c)$. This way, since tokens can be publicly stored, each consumer c will have to manage a single key k_c only and compute, when needed for accessing a resource $r \in \mathsf{cap}(c)$, the encryption key k_r.

While effective, this simple solution might create more tokens than necessary. Consider two consumers ε and ζ such that $\mathsf{cap}(\varepsilon) = \mathsf{cap}(\zeta) = \mathtt{fgh}$. To permit both consumers to access the three resources, six tokens (three for permitting derivation of $k_{\mathtt{f}}$, $k_{\mathtt{g}}$, and $k_{\mathtt{h}}$ starting from k_ε, and three for permitting the same derivation starting from k_ζ) would be needed, as illustrated in Figure 3(a). The insertion of an intermediate vertex (i.e., key) in the DAG modeling key derivation would still permit the same derivations, while saving a token, as illustrated in Figure 3(b): it is easy to see that both ε and ζ are still able to derive all (and

only) the keys they are entitled to with the creation of five (rather than six) tokens. Such additional keys are used for derivation purposes only, and are not used to encrypt any resource (nor are they assigned to any consumer). They can be defined according to different criteria: in traditional cloud-based scenarios they are typically associated with sets of users [5]. Considering the peculiarities of the market scenario, and the fact that consumers dynamically join and leave the market to acquire sets of resources, it is natural to associate them with sets of resources. We illustrate our key derivation structure in the remainder of this section.

4.2 Key Derivation Structure and Correctness

A *key derivation structure* for our problem is formally defined as follows [6].

Definition 1 (Key derivation structure). *Let $\mathcal{R} = \{r_1, \ldots, r_n\}$ be a set of resources and $\mathcal{C} = \{c_1, \ldots, c_m\}$ be a set of consumers. A* key derivation structure *over $\mathcal{R}$ and $\mathcal{C}$ is a DAG G(V,E) such that:*

1. $\forall v_x \in V$, $(x \in \mathcal{C}) \vee (x \subseteq \mathcal{R})$;
2. $\forall c \in \mathcal{C}$, $v_c \in V$;
3. $\forall r \in \mathcal{R}$, $v_r \in V$;
4. $\forall (v_x, v_y) \in E : (y \subset x) \vee (x \in \mathcal{C} \wedge y \subseteq \mathcal{R})$.

The definition above formally characterizes a generic key derivation structure for a data market over a set of resources and a set of consumers. Condition 1 states that a key derivation structure includes a set of vertices representing consumers or sets of resources. Condition 2 demands that, for each consumer c, a key derivation structure includes a vertex v_c for c. Similarly, Condition 3 demands that, for each resource r, a key derivation structure includes a vertex v_r for r. Lastly, Condition 4 states that the (directed) edges of a key derivation structure connect either a consumer to a set of resources ($x \in \mathcal{C} \wedge y \subseteq \mathcal{R}$), or a set of resources to a subset of the same ($y \subset x$). Recall (Section 3.1) that vertices in a key derivation structure represent encryption keys. Each consumer $c \in \mathcal{C}$ knows the encryption key k_c represented by vertex v_c. Similarly, each resource $r \in \mathcal{R}$ is encrypted with the key k_r represented by vertex v_r.

Example 2. Figure 4 illustrates an example of key derivation structure defined over the four consumers $\alpha, \beta, \gamma, \delta$ and five resources $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}$ of our running example (Example 1). The structure includes a vertex for each consumer (denoted, for readability, with a gray background), a vertex for each resource, and two additional vertices defined over sets of resources. As already noted, for simplicity in the figures we denote each vertex v_x with x (e.g., $\mathtt{a}$ is the vertex for resource $\mathtt{a}$ representing the encryption key $k_{\mathtt{a}}$ used to encrypt it and, similarly, α is the vertex for consumer α representing her encryption key k_α).

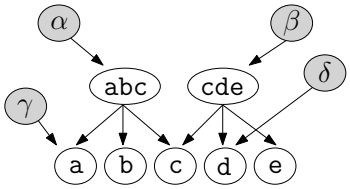


Fig. 4. An example of a key derivation structure enforcing the authorizations in Figure 2

The definition of a key derivation structure creates (direct and/or indirect) derivation relationships among encryption keys (be them assigned to consumers, to resources, or to group of resources). Clearly, such derivation relationships must correctly enforce the authorization policy so to ensure that each consumer can derive all and only the encryption keys used for encrypting the resources she can access. This requires to guarantee that each consumer c can reach, starting from the vertex v_c representing her encryption key k_c, all and only the vertices representing the encryption keys k_r used to encrypt the resources in her capability list $\mathsf{cap}(c)$. Formally, given an authorization policy $\mathcal{A}$ over $\mathcal{R}$ of resources and $\mathcal{C}$, a key derivation structure *correctly enforces* $\mathcal{A}$ iff $\forall r{\in}\mathcal{R}, \forall c{\in}\mathcal{C}$ it holds that $r{\in}\mathsf{cap}(c) \Leftrightarrow \exists$ a path in G from v_c to v_r.

Example 3. Consider the authorization policy expressed by the capability lists reported in Figure 2. The key derivation structure in Figure 4 correctly enforces the policy. The structure includes one vertex for each resource, one vertex for each consumer, and two additional vertices representing the non-singleton capability lists of consumers α ($\mathsf{cap}(\alpha) = \mathtt{abc}$) and β ($\mathsf{cap}(\beta) = \mathtt{cde}$). The edges in the structure connect, linking their vertices, each consumer to her capability list, and sets of resources respecting subset containment relationship. It is easy to see that each consumer can reach through a path all and only the resources in her capability list, meaning she can derive the correct encryption keys. For example, consumer α can, from her key k_α and with token $t_{\alpha,\mathtt{abc}}$, derive $k_{\mathtt{abc}}$, from which she can use tokens $t_{\mathtt{abc,a}}$, $t_{\mathtt{abc,b}}$, and $t_{\mathtt{abc,c}}$ to derive $k_{\mathtt{a}}$, $k_{\mathtt{b}}$, and $k_{\mathtt{c}}$.

Given an authorization policy, a correct key derivation structure can be defined in different ways, depending on how the additional vertices (i.e., those not corresponding to consumers nor to resources, and used for derivation purposes only) are defined and how edges are defined to connect vertices enabling key derivation. Our approach consists in including in the structure a vertex for each capability list of consumers in $\mathcal{C}$, and an edge $(v_c, v_{\mathsf{cap}(c)})$ connecting each consumer c to her capability list $\mathsf{cap}(c)$. We then include edges connecting vertices representing sets of resources (e.g., capability lists) ensuring that the set R represented by vertex v_R is *covered* guaranteeing that the set of resources represented by the vertices directly reachable from v_R is equal to R. The rationale behind this approach is quite intuitive: each consumer c can, starting from her

PUBLISH_RESOURCE(r,$G(V,E)$)

1: generate key k_r for r
2: generate label l_r for r
3: generate vertex v_r for r
4: $V := V \cup \{v_r\}$
5: $enc\_r :=$ **encrypt**(r,k_r)
6: **publish**(enc_r)

Fig. 5. Management of the publication of a resource r

own key (vertex v_c, included in the structure by Condition 2 in Definition 1), derive the key associated with her capability list (vertex $v_{\mathsf{cap}(c)}$ in the structure). Since $\mathsf{cap}(c)$ is covered, and since (by Condition 3 in Definition 1) the structure includes a vertex for each resource in $\mathsf{cap}(c)$, c can derive (directly and/or indirectly) all and only the keys for all resources in $\mathsf{cap}(c)$. For example, the key derivation structure in Figure 4 is built following this approach, and correctly enforces the authorizations in Figure 2. In the following sections, we illustrate our approach for managing the publication of resources on the market, and for enforcing access restrictions through the definition and maintenance of a key derivation structure.

5 Resource Management

An empty key derivation structure $G(V,E)$ is created whenever an owner interacts for the first time with the data market. Such key derivation is then updated whenever a resource is published by the owner on the market, as well as whenever consumers are granted access to resources, to guarantee its correctness and the correctness of the key derivations enabled. In this section, we illustrate our approach for managing the publication of a resource. We will discuss how the purchase of a resource by a consumer can be managed in Section 6.

Figure 5 illustrates the pseudocode of procedure **Publish_Resouce**, managing the publication of a new resource on the market. The procedure takes as input a resource r and an existing key derivation structure $G(V,E)$, possibly empty if r is the first resource to be published by the owner. The procedure generates an encryption key k_r, which will be used for encrypting r, and the corresponding label l_r, used for enabling derivation (lines 1–2). It then creates the corresponding vertex v_r, and inserts v_r into V (lines 3–4). Resource r is encrypted with key k_r (line 4), and the resulting encrypted version enc_r is published on the market (line 6).

Example 4. Consider the publication of the five resources of our running example (Example 1). Figure 6 illustrates an example of a key derivation structure after the publication on the market of the resources. Since no authorizations have been granted yet, the structure includes only one vertex for each resource, and no vertex for consumers nor sets of resources.

Fig. 6. An example of a key derivation structure after publishing the five resources of Example 1, including one vertex for each resource

6 Access Control Enforcement

The key derivation structure is updated to reflect purchases by consumers, to ensure that the key derivation enabled by the tokens represented by edges in the structure correctly enforces the granted authorizations. As mentioned in Section 4, we recall that revocation of an access already granted is not in line with the peculiarities of the considered market scenario. Clearly, this does not imply that once an owner has published a resource on the market she cannot remove it: as a matter of fact, owners remain in full control of their resources and, should they wish so, they can always remove resources from the market. Issues that may arise if owners remove resources for which consumers have paid are beyond the scope of this paper. We illustrate in Section 6.1 how to update a key derivation structure aiming at minimizing the size of the entailed token catalog (thus ensuring fast access to tokens by consumers), and an alternative strategy aiming at minimizing the number of additional tokens (thus reducing the costs entailed by updating the catalog stored on-chain).

6.1 Reducing token catalog size

Figure 7 illustrates the pseudocode of procedure **Min_Token_Catalog**, updating a key derivation structure to accommodate –and reflect in the key derivations enabled– the purchase of a set R of resources by a consumer c, aiming at minimizing the overall number of tokens. The procedure takes as input the consumer c, her current capability list $\mathsf{cap}(c)$, the set R of resources for which c has paid a reward to the owner and for which the owner grants access, and the key derivation structure $G(V,E)$ to be updated. It updates the structure to enable c to derive the keys needed to decrypt the resources in $\mathsf{cap}(c) \cup R$.

The procedure first checks whether the structure already contains a vertex for c, meaning that c has already interacted with the market and the owner, and has her own key k_c. If this is not the case, the procedure generates a key k_c, a corresponding vertex v_c, and adds v_c to the structure (lines 1–4). Otherwise, the procedure removes the edge connecting v_c to vertex $v_{\mathsf{cap}(c)}$, representing her current (meaning before the purchase of R) capability list, and the corresponding token from the catalog (lines 5–7). The procedure then checks whether it is possible to remove from the structure $v_{\mathsf{cap}(c)}$ and, if so, whether removing its incident edges and directly connecting its ancestors to its descendants can reduce the number of tokens in the catalog (edges in the structure, lines 8–18). To do so, it checks whether other consumers have the same current capability list as,

MIN_TOKEN_CATALOG(c,$\mathsf{cap}(c)$,R,$G(V,E)$)

1: **if** $v_c \notin V$ **then**
2: generate key k_c for c
3: generate vertex v_c for c
4: $V := V \cup \{v_c\}$
5: **else**
6: $E := E \setminus \{(v_c, v_{\mathsf{cap}(c)})\}$
7: $\mathcal{T} := \mathcal{T} \setminus \{t_{c,\mathsf{cap}(c)}\}$
8: **if** $\nexists c' \neq c$ s.t. $\mathsf{cap}(c) = \mathsf{cap}(c')$ **then**
9: let Anc be the ancestors of $v_{\mathsf{cap}(c)}$
10: let $Desc$ be the descendants of $v_{\mathsf{cap}(c)}$
11: **if** $(|Anc| \times |Desc|) < (|Anc| + |Desc|)$ **then**
/* removing $v_{\mathsf{cap}(c)}$ and its incident edges reduces the number of tokens */
12: **for each** $v_a \in Anc$ **do**
13: $E := E \setminus \{(v_a, v_{\mathsf{cap}(c)})\}$
14: $\mathcal{T} := \mathcal{T} \setminus \{t_{a,\mathsf{cap}(c)}\}$
15: **for each** $v_d \in Desc$ **do**
16: $E := E \setminus \{(v_{\mathsf{cap}(c)}, v_d)\} \cup \{(v_a, v_d)\}$
17: $\mathcal{T} := \mathcal{T} \setminus \{t_{\mathsf{cap}(c),d}\} \cup \{t_{a,d}\}$
18: $V := V \setminus \{v_{\mathsf{cap}(c)}\}$
19: $\mathsf{cap}(c) := \mathsf{cap}(c) \cup R$
20: **if** $v_{\mathsf{cap}(c)} \in V$ **then**
21: $E := E \cup \{(v_c, v_{\mathsf{cap}(c)})\}$
22: $\mathcal{T} := \mathcal{T} \cup \{t_{c,\mathsf{cap}(c)}\}$
23: **else**
24: generate key $k_{\mathsf{cap}(c)}$ for $\mathsf{cap}(c)$
25: generate label $l_{\mathsf{cap}(c)}$ for $\mathsf{cap}(c)$
26: generate vertex $v_{\mathsf{cap}(c)}$ for $\mathsf{cap}(c)$
27: $V := V \cup \{v_{\mathsf{cap}(c)}\}$
28: $E := E \cup \{(v_c, v_{\mathsf{cap}(c)})\}$
29: $\mathcal{T} := \mathcal{T} \cup \{t_{c,\mathsf{cap}(c)}\}$
/* connect $v_{\mathsf{cap}(c)}$ ensuring that $\mathsf{cap}(c)$ is covered */
30: $CandCover := \{v_X \in V : X \subseteq \mathsf{cap}(c)\}$
31: $Cover := \emptyset$
32: $ToCover := \mathsf{cap}(c)$
33: **while** $ToCover \neq \emptyset$ **do**
34: let $v_Y \in CandCover$ be the vertex s.t. $Y \cap ToCover$ is largest
35: $Cover := Cover \cup \{v_Y\}$
36: $CandCover := CandCover \setminus \{v_Y\}$
37: $ToCover := ToCover \setminus Y$
38: $E := E \cup \{(v_{\mathsf{cap}(c)}, v_Y)\}$
39: $\mathcal{T} := \mathcal{T} \cup \{t_{\mathsf{cap}(c),Y}\}$
/* check if inserting $v_{\mathsf{cap}(c)}$ as an intermediate vertex can reduce the number of tokens */
40: let $Par \subseteq V$ be the set of vertices over a set of resources $\supset \mathsf{cap}(c)$
41: let $ReachCover$ be the set of vertices reachable from vertices in $Cover$
42: **for each** $v_{par} \in Par$ **do**
43: $ToRemove := \emptyset$
44: **for each** $v \in ReachCover \cup Cover$ **do**
45: **if** $(v_{par}, v) \in E$ **then**
46: $ToRemove := ToRemove \cup \{(v_{par}, v)\}$
47: **if** $|ToRemove| \geq 2$ **then**
48: $E := E \cup \{(v_{par}, v_{\mathsf{cap}(c)})\}$
49: $\mathcal{T} := \mathcal{T} \cup \{t_{par,\mathsf{cap}(c)}\}$
50: **for each** $(v_{par}, v_z) \in ToRemove$ **do**
51: $E := E \setminus \{(v_{par}, v_z)\}$
52: $\mathcal{T} := \mathcal{T} \setminus \{t_{par,z}\}$

Fig. 7. Management of purchases reducing the size of the token catalog

clearly, if this is the case, then the vertex could not be removed (line 8). It determines the sets Anc and $Desc$ of ancestors and descendants of $v_{\mathsf{cap}(c)}$ (lines 9–10).

If $(|Anc| \times |Desc|) < (|Anc| + |Desc|)$ the procedure removes all $v_{\mathsf{cap}(c)}$'s incident edges and connects all $v_{\mathsf{cap}(c)}$'s ancestors to $v_{\mathsf{cap}(c)}$'s descendants, removing also the corresponding tokens from the catalog (lines 11–17). The procedure removes $v_{\mathsf{cap}(c)}$ from the structure (line 18) and updates c's capability list to $\mathsf{cap}(c) \cup R$ (line 19). If the structure already includes a vertex for the updated capability list, the procedure simply adds an edge (and the corresponding token) connecting v_c to the vertex, and terminates (lines 20–22). Otherwise (line 23), it generates a key for $\mathsf{cap}(c)$, the corresponding label and vertex, and inserts the vertex in the structure (lines 24–27), also connecting v_c to it through an edge, and adding the corresponding token to the catalog (lines 28–29). To guarantee correctness of the key derivation, the procedure then connects the newly created vertex $v_{\mathsf{cap}(c)}$ to other vertices of the structure, ensuring coverage (i.e., ensuring that the vertex is connected to other vertices so that the union of the resources of the vertices directly reachable from $v_{\mathsf{cap}(c)}$ is equal to $\mathsf{cap}(c)$, lines 30–39). It does so defining three sets *CandCover*, *Cover*, and *ToCover*, modeling respectively the vertices that are candidate for coverage (i.e., for becoming direct descendants of $v_{\mathsf{cap}(c)}$), the vertices that are actually selected for coverage, and the resources to be covered. Such sets are initialized, respectively, to the set of vertices in the structure defined over a subset of $\mathsf{cap}(c)$ (line 30), to an empty set (line 31), and to $\mathsf{cap}(c)$ (line 32). The procedure heuristically selects, among the candidates in *CandCover*, the vertex v_Y that covers the largest number of the resources to be covered in *ToCover* (i.e., such that $Y \cap$ *ToCover* is largest, lines 34–35), removes it from the candidates (line 36), removes the set Y of resources from *ToCover* (line 37), and connects $v_{\mathsf{cap}(c)}$ to v_Y adding the corresponding token to the catalog (lines 38–39). These operations are repeated until all resources in $\mathsf{cap}(c)$ have been covered (i.e., *ToCover* is empty, line 33). Finally, the procedure checks whether the overall number of tokens can be further reduced thanks to the creation of $v_{\mathsf{cap}(c)}$ [4]. The procedure first identifies a set *Par* of vertices representing supersets of $\mathsf{cap}(c)$, and a set *ReachCover* of vertices reachable from the vertices used for coverage (i.e., those in *Cover*) (lines 40–41). To this aim, if a vertex v_{par} in *Par* is directly connected to more than one vertex in *Cover* and/or in *ReachCover*, the procedure inserts $v_{\mathsf{cap}(c)}$ as an intermediate vertex, and removes the edges from v_{par} to the vertices in *Cover* $\cup$ *DescCover* (lines 42–52), reducing the number of edges (and of corresponding tokens).

Example 5. Consider the key derivation structure in Figure 4. Figure 8 illustrates the evolution of the structure to enforce a sequence of requests. The structure in Figure 8(a) grants γ access to `b` and `c`: the token enabling the derivation of k_{a} from k_γ (modeled by edge $(v_\gamma, v_{\mathsf{a}})$) is replaced with a token enabling the derivation of k_{abc} from k_γ (modeled by edge $(v_\gamma, v_{\mathsf{abc}})$). Similarly, the structure in Figure 8(b) grants β access to `a` and `b`, starting from the structure in Figure 8(a): vertex v_{abcde}, representing $\mathsf{cap}(\beta) = \mathtt{abcde}$, is inserted in the structure and is connected to vertices v_{abc} and v_{cde} (left-hand-side structure). Following this purchase, vertex v_{cde} becomes redundant, and is removed, saving two tokens while permitting the same derivations (right-hand-side structure). Finally, Figure 8(c) grants α access to `e`, starting from the structure in Figure 8(b): vertex

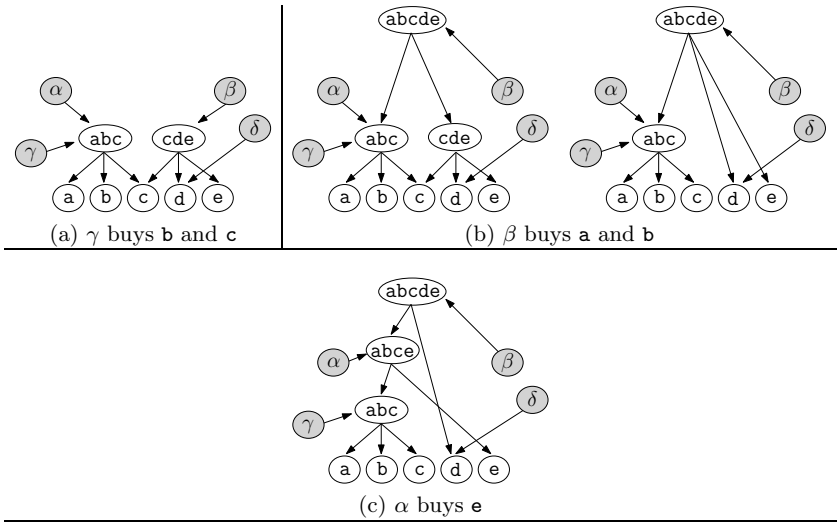


Fig. 8. Evolution of the key derivation structure in Figure 4 to enforce a sequence of purchases minimizing the overall number of tokens

$v_{\mathtt{abce}}$ is inserted in the structure, v_α is connected to it, and our approach places it also as direct descendant of $v_{\mathtt{abcde}}$, saving one token in the catalog.

6.2 Reducing token catalog updates

The approach illustrated in the previous section aims to limit the size of the token catalog by deleting the tokens that become unnecessary after accommodating the purchase of R by a consumer c (e.g., the token represented by the edge connecting v_c to $v_{\mathsf{cap}(c)}$, with $\mathsf{cap}(c)$ the previous capability list of c, which becomes redundant due to the fact that v_c will be connected to vertex $v_{\mathsf{cap}(c)\cup R}$ and thanks to the coverage of $\mathsf{cap}(c)\cup R$), and by re-organizing the key derivation structure, possibly deleting unnecessary vertices (e.g., $v_{\mathsf{cap}(c)}$) and deleting old tokens and adding new tokens leveraging the deletion of $v_{\mathsf{cap}(c)}$ and the inclusion of $v_{\mathsf{cap}(c)\cup R}$. All these operations have clearly a cost, and can represent an issue as they are reflected on the blockchain as token catalog is stored on-chain [6]. To limit the update (write) operations on the blockchain, we propose an alternative approach that: *i)* does not require token deletion; and *ii)* tries to re-use as much as possible elements already included in the key derivation structure.

This approach follows the observation that token deletion is not necessary for enforcing new access grants, and re-using elements already existing in the structure limits the number of the tokens added to accommodate the policy update. To illustrate, consider the purchase of a set R of resources by consumer c. Indeed, the capability list of c needs to be updated to include the resources in R (i.e., it is to be updated to $\mathsf{cap}(c \cup R)$) and, to ensure correctness of key

derivation, v_c must be connected with a vertex in the structure representing $\mathsf{cap}(c) \cup R$. If such a vertex already belongs to the hierarchy, it is possible to add to the structure an edge $(v_c, v_{\mathsf{cap}(c)\cup R})$, representing the token enabling the derivation of $k_{\mathsf{cap}(c)\cup R}$ from k_c, without deleting the (now redundant, but not incorrect) edge $(v_c, v_{\mathsf{cap}(c)})$ and the corresponding token. This increases the overall number of tokens, but reduces the operations performed to accommodate the purchase. If, otherwise, $v_{\mathsf{cap}(c)\cup R}$ does not belong to the hierarchy, two different approaches can be adopted.

- *Insertion of a new vertex*: insert vertex $v_{\mathsf{cap}(c)\cup R}$ in the structure, and connect it to $v_{\mathsf{cap}(c)}$ (and hence indirectly to the corresponding resources) and, directly or indirectly, to the vertices of the resources in R;
- *Relabeling of an existing vertex*: connect (directly or indirectly) vertex $v_{\mathsf{cap}(c)}$ to the vertices of the resources in R, relabeling $v_{\mathsf{cap}(c)}$ to represent $\mathsf{cap}(c)\cup R$. Compared to the previous strategy, this strategy saves the insertion and deletion of one token (the one from v_c to $v_{\mathsf{cap}(c)\cup R}$, as it already exists), but it is not always viable. In particular, relabeling $v_{\mathsf{cap}(c)}$ to $v_{\mathsf{cap}(c)\cup R}$ can be adopted only if: *i)* no other consumer c' still has a capability list equal to the (previous) list of c (i.e., $\mathsf{cap}(c')=\mathsf{cap}(c)$), as otherwise c' would, as a consequence of the connection of $v_{\mathsf{cap}(c)}$ with the vertices for R, gain access to R without being authorized; and *ii)* all the ancestors of the relabeled vertex represent a superset of $\mathsf{cap}(c) \cup R$, since otherwise consumers who can derive such vertices would be granted access to R.

Independently from the strategy chosen for including vertex $v_{\mathsf{cap}(c)\cup R}$ in the structure, such vertex needs to be connected to the resources in R (as it is clearly already connected to those in $\mathsf{cap}(c)$). If the structure already contains a vertex v_R, it is sufficient to add to the structure edge $(v_{\mathsf{cap}(c)\cup R}, v_R)$ and define the corresponding token. If this is not the case, a possible solution -always available- can consist in adding $|R|$ edges, one for each resource in R. However, if the structure already includes a set K of less than $|R|$ vertices representing subsets of $\mathsf{cap}(c)\cup R$ that completely cover R, the number of additional edges (and hence of the tokens to be added) can be reduced to $|K|$ by connecting $\mathsf{cap}(c) \cup R$ with the vertices in K.

Figure 9 illustrates the pseudocode of procedure **Min_Additional_Tokens**, updating a key derivation structure to accommodate –and reflect in the key derivations enabled– the purchase of a set R of resources by a consumer c, aiming at minimizing the number of additional tokens. The procedure takes as input the consumer c, her current capability list $\mathsf{cap}(c)$, the set R of resources for which c has paid a reward to the owner and for which the owner grants access, and the key derivation structure $G(V,E)$ to be updated. It updates the structure to enable c to derive the keys needed to decrypt the resources in $\mathsf{cap}(c)\cup R$.

The procedure first checks whether the structure already contains a vertex for c, meaning that c has already interacted with the market and the owner, and has her own key k_c. If this is not the case, the procedure generates a key k_c, a corresponding vertex v_c, and adds v_c to the structure (lines 1–4). The procedure then

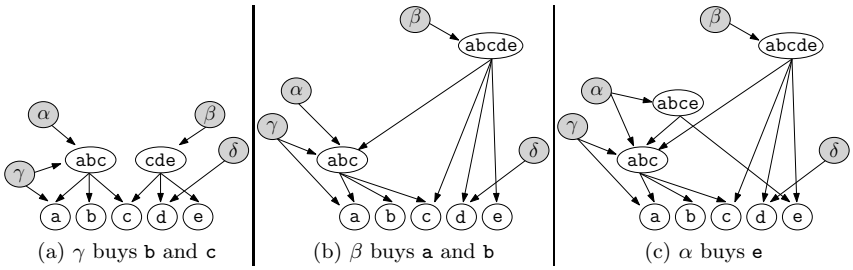
MIN_ADDITIONAL_TOKENS(c,$\mathsf{cap}(c)$,R,$G(V,E)$)

1: **if** $v_c \notin V$ **then**
2: generate key k_c for c
3: generate vertex v_c for c
4: $V := V \cup \{v_c\}$
5: **if** $v_{\mathsf{cap}(c)\cup R} \in V$ **then**
6: $E := E \cup \{(v_c, v_{\mathsf{cap}(c)\cup R})\}$
7: $\mathcal{T} := \mathcal{T} \cup \{t_{c,\mathsf{cap}(c)\cup R}\}$
8: **else**
9: **if** $\nexists c' \neq c \in \mathcal{C} : \mathsf{cap}(c') = \mathsf{cap}(c)$ **then**
10: let Anc be the ancestors of $v_{\mathsf{cap}(c)}$
11: **if** $\forall v_X \in Anc : X \supset \mathsf{cap}(c) \cup R$ **then**
12: relabel $v_{\mathsf{cap}(c)}$ into $v_{\mathsf{cap}(c)\cup R}$
13: **else**
14: generate key $k_{\mathsf{cap}(c)\cup R}$
15: generate label $l_{\mathsf{cap}(c)\cup R}$
16: generate vertex $v_{\mathsf{cap}(c)\cup R}$
17: $V := V \cup \{v_{\mathsf{cap}(c)\cup R}\}$
18: $E := E \cup \{(v_{\mathsf{cap}(c)\cup R}, v_{\mathsf{cap}(c)})\}$ /* cover $\mathsf{cap}(c)$ */
19: $\mathcal{T} := \mathcal{T} \cup \{t_{\mathsf{cap}(c)\cup R,\mathsf{cap}(c)}\}$
/* connect $v_{\mathsf{cap}(c)\cup R}$ ensuring that R is covered */
20: $CandCover := \{v_X \in V : X \subseteq \mathsf{cap}(c) \cup R\}$
21: $ToCover := R$
22: **while** $ToCover \neq \emptyset$ **do**
23: let $v_Y \in CandCover$ be the vertex s.t. $Y \cap ToCover$ is largest
24: $CandCover := CandCover \setminus \{v_Y\}$
25: $ToCover := ToCover \setminus Y$
26: $E := E \cup \{(v_{\mathsf{cap}(c)\cup R}, v_Y)\}$
27: $\mathcal{T} := \mathcal{T} \cup \{t_{\mathsf{cap}(c)\cup R,Y}\}$
28: $E := E \cup \{(v_c, v_{\mathsf{cap}(c)\cup R})\}$
29: $\mathcal{T} := \mathcal{T} \cup \{t_{c,\mathsf{cap}(c)\cup R}\}$

Fig. 9. Management of purchases reducing the number of additional tokens

checks whether the structure already includes a vertex for $\mathsf{cap}(c) \cup R$ (line 5). If this is the case, the procedure simply adds an edge connecting v_c to $v_{\mathsf{cap}(c)\cup R}$, creates the corresponding token, and terminates (lines 5–7). Otherwise, it checks whether it is possible to relabel the vertex $v_{\mathsf{cap}(c)}$: to this end, it checks whether no consumer c' has capability list equal to $\mathsf{cap}(c)$ (line 9), and whether all ancestors of $v_{\mathsf{cap}(c)}$ are defined over supersets of $\mathsf{cap}(c) \cup R$ (lines 10-11). If so, the procedure relabels $v_{\mathsf{cap}(c)}$ into $v_{\mathsf{cap}(c)\cup R}$ (line 12), resulting in the structure to include a vertex for $\mathsf{cap}(c) \cup R$. If this is not the case, the procedure generates an encryption key and a label for $\mathsf{cap}(c) \cup R$, creates the corresponding vertex, adds it to the structure (lines 13–17), and connects it to $v_{\mathsf{cap}(c)}$ (lines 18–19). The procedure then covers the resources in R starting from the vertices already included in the structure that have the larges intersection with R, to reduce the number of needed connections (i.e., of added edges, hence of additional tokens, lines 20–27). Finally, the procedure connects v_c to vertex $v_{\mathsf{cap}(c)\cup R}$ representing $\mathsf{cap}(c) \cup R$, creating the corresponding token (lines 28–29).

Example 6. Consider the key derivation structure in Figure 4. Figure 10 illustrates the evolution of the structure to enforce the same sequence of requests as in Figure 8, but minimizing the number of additional tokens. The structure in Figure 10(a) grants γ access to `b` and `c`: since the structure already includes



(a) γ buys `b` and `c` (b) β buys `a` and `b` (c) α buys `e`

Fig. 10. Evolution of the key derivation structure in Figure 4 to enforce a sequence of purchases minimizing the number of additional tokens

a vertex $v_{\texttt{abc}}$ for the updated capability list of γ, the structure is updated simply adding a token enabling the derivation of $k_{\texttt{abc}}$ from k_γ (modeled by edge $(v_\gamma, v_{\texttt{abc}})$): the (now redundant) edge $(v_\gamma, v_{\texttt{a}})$ is left untouched, saving on its deletion. The structure in Figure 10(b) grants β access to `a` and `b`, starting from the structure in Figure 10(a): vertex $v_{\texttt{cde}}$ is relabeled as $v_{\texttt{abcde}}$, and is covered by connecting it to $v_{\texttt{abc}}$, which completely covers resources `ab` purchased by β. Note that this approach implies the simple insertion of one token (modeled by edge $(v_{\texttt{abcde}}, v_{\texttt{abc}})$), in contrast to the insertion of 4 tokens and the removal of 4 tokens paid by the solution minimizing the token catalog size in Figure 8(b). Finally, Figure 10(c) grants α access to `e`, starting from the structure in Figure 10(b): in this case it would not be possible to relabel $v_{\texttt{abc}}$ as $v_{\texttt{abce}}$, as this would also allow γ to access `e`. In this case, it is then necessary to insert a new vertex $v_{\texttt{abce}}$, connected to $v_{\texttt{abc}}$ and $v_{\texttt{e}}$. Again, the (now redundant) edge $(v_\alpha, v_{\texttt{abc}})$ is left untouched, avoiding its deletion.

6.3 Further optimizations

The reduction of the additional tokens introduced to accommodate resource purchase could leverage further optimizations, which we illustrate in the remainder of this section.

- **Remove the assumption of having a node representing the capability list of each customer.** The two strategies we have illustrated build on the inclusion in the hierarchy of a vertex representing the capability list of the consumers. This requires the creation of a new vertex, or the relabeling of an existing one, at each purchase of a resource. Removing this assumption may reduce the number of tokens that need to be created to manage a purchase operation. To illustrate, consider a consumer c who had purchased a set of resources, and suppose she is now purchasing a set R of resources. Suppose that the key derivation structure already includes a vertex for R. In principle, to enforce such purchase, it could be sufficient add an edge to the structure linking v_c to v_R, hence requiring the insertion of a single token in the catalog. Requiring to have, in the structure, vertex $v_{\texttt{cap}(c)\cup R}$

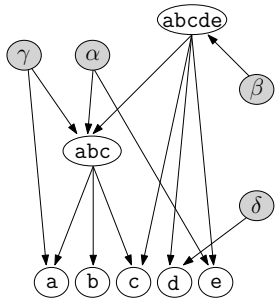


Fig. 11. Evolution of the key derivation hierarchy in Figure 4 removing the assumption of having all capability vertices

would instead imply the insertion of at least two edges (hence two tokens): one connecting v_c to $v_{\mathsf{cap}(c)\cup R}$, one connecting $v_{\mathsf{cap}(c)\cup R}$ to v_R. Figure 11 illustrates the key derivation structure obtained, starting from the structure in Figure 4 and managing the purchases in Figure 10, according to this optimization strategy. In particular, the purchase of `e` by α is enforced by connecting v_α directly to $v_{\mathtt{e}}$, saving 3 tokens w.r.t. Figure 10. Note that we did not manage the purchase of `a` and `b` by β and of `b` and `c` by γ connecting directly their vertices to the resources' vertices, since this would not provide any cost reduction.

– **Manage purchases in batch.** The two strategies we have illustrated assume that the owner manages each request, independently from other ones, as soon as it is submitted by a consumer. However, it can be noted that the combined management of multiple requests together may enable considerable benefits. A first intuitive advantage relates to allowing the owner to be not available all the time for managing resource purchase: in agreement with the consumers (and possibly with adjustments to the reward to be paid for the resources), the owner may opt for a periodic (e.g., daily, or after a certain number of requests have been submitted) re-organization of the key derivation structure. Also, when some requests by different consumers in a batch relate to the same resources, there can be a saving in the number of tokens needed to accommodate them. To illustrate, consider the four consumers and five resources of our running example, and suppose a sequence of access requests such that, starting from the structure in Figure 4, *i)* `a` is requested by β, *ii)* `c` is requested by γ, *iii)* `b` is requested by β, *iv)* `e` is requested by α, and *v)* `b` is requested by γ. Figure 12(a) illustrates the key derivation structure obtained managing the sequence in the order, updating the structure at every request. Figure 12(b) illustrates the key derivation structure obtained managing the same purchases in batch. While the first strategy implies the addition of 8 tokens, the second one requires only 6 additional tokens (note that, for readability, in these structures of Figure 12

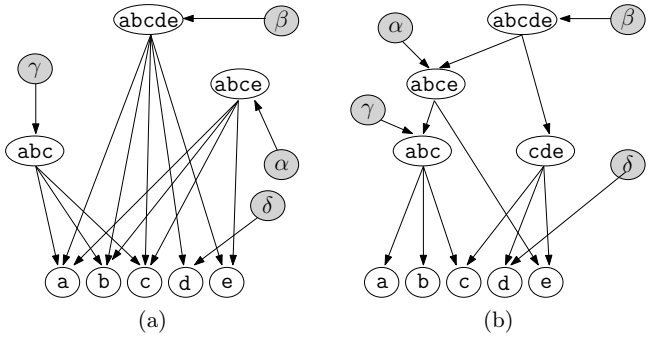


Fig. 12. Evolution of the key derivation hierarchy in Figure 4 managing purchases in order (a) and in batch (b)

we did not report the tokens that became redundant due to the enforcement of the new access grants).

– **Add additional vertices**. A third possibility aiming at further reducing the number of additional tokens can leverage the observation that the insertion of additional nodes referring to sets of resources at a certain point in time may reduce the number of tokens that will be needed in the future, to accommodate future requests. The presence of a vertex in the structure representing a set R of resources could be profitably used for accommodating another purchase for a superset of R, with a saving in the number of tokens. Intuitively, if a set $R' \supset R$ of resources is acquired by at least two different consumers, the presence of R in the structure can reduce the number of tokens needed to accommodate such requests. A possible strategy following this intuition is to materialize (i.e., create and insert into the structure) the vertex representing the set R of resources acquired by a consumer. The data owner clearly pays an additional token for inserting R, but she may experience a saving in the future. To illustrate, consider the key derivation structure in Figure 4. To enforce the purchase of resources `a` and `b` by β, the owner might materialize a vertex $v_{\texttt{ab}}$ (though it does not correspond to any capability list). Suppose that, after some time, also consumers δ and ε request access to these resources. Having $v_{\texttt{ab}}$ in the structure, it is sufficient to insert a token from $v_{\texttt{abd}}$ to $v_{\texttt{ab}}$ for δ, and from v_ϵ to $v_{\texttt{ab}}$ for ε, saving on the number of tokens inserted to manage the three purchases (7 tokens instead of 9 tokens). Figure 13 illustrates the resulting key derivation structure.

The optimizations illustrated above, while helping in further reducing the number of additional tokens, require analysis to balance the benefits against some complications they entail. In particular, for the first strategy, we note that the availability of vertices representing groups of resources in the structure can be beneficial to accommodate future requests (possibly by other consumers) which can be enforced leveraging such vertices. This reasoning clearly applies also to

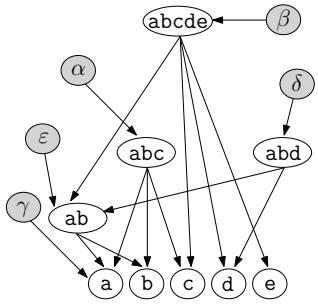


Fig. 13. Evolution of the key derivation hierarchy in Figure 4 inserting additional vertices

vertices for capability lists. Removing the assumption of having them in the hierarchy can be beneficial to enforce a specific purchase, but may result –in the long run– in additional tokens to be then added to accommodate future requests. Also, if adopted from initialization time (i.e., from the very first request), it creates a structure with vertices for consumers and for single resources, resulting in a bipartite graph where each consumer c is directly connected by $|\mathsf{cap}(c)|$ edges (with an equal number of tokens in the catalog) to each resource in her capability list. For the second strategy (managing batches of purchases), it can clearly help in optimizing the key derivation structure, but brings the inevitable drawback of introducing delays in making resources available to consumers. As for the last strategy, additional vertices can be beneficial if they can be used to accommodate future requests, which requires some analysis by the owner to forecast, in a reliable way, future requests.

7 Related Work

The adoption of selective encryption for enforcing access restrictions in digital data markets, coupled with smart contracts deployed on a blockchain, has first been proposed in [6]. Our solution builds on this proposal and on the use of selective encryption and key derivation for enabling data owners to maintain control over their resources in the data market to propose different optimization strategies for effectively updating the key derivation structure enforcing new access restrictions. A preliminary version of this work appeared in [8], which is here extended with more complete and revised algorithms, and enhanced discussions on the possible strategies that can be adopted along with their pros and cons.

The adoption of selective encryption, possibly combined with key derivation, has been widely adopted as a means for protecting data in outsourcing scenarios, which are characterized by data owners storing their resources on the premises of non fully trusted (cloud) providers (e.g., [2, 4, 5]), and which would benefit from data self-enforcing access restrictions through an encryption layer. These approaches however operate in a different scenario and aim at enforcing a (quite

static) authorization policy defined by the data owner. The key derivation hierarchy is organized to model the access control lists of resources (i.e., nodes represent groups of users), in contrast to capability lists. Changes in the authorization policy can imply both grant and revoke of privileges as decided by the data owner, who is interested in limiting her intervention to enforce policy updates. Also, they do not consider the peculiarities of digital data markets, such as the payment of rewards to the owner, nor the integration with blockchain and smart contracts.

Other lines of work close to ours are related to the adoption of blockchain and smart contracts for data management and access control (e.g., [7, 9–17]). These approaches are however complementary to ours, as they do not consider selective encryption and key derivation for enforcing access restrictions to traded resources or do not consider the peculiarities of data markets.

8 Conclusions

We investigated the problem of maintaining owners in control over their data when shared and traded in digital data markets through selective owner-side encryption and smart contracts on a blockchain. We proposed different approaches for enforcing selective access restrictions by properly modifying the key derivation structure and the catalog of tokens used to distribute keys to authorized subjects. Our strategies pursue different optimization criteria that suit the peculiarities of the digital data market scenario, and aim at reducing the size of the token catalog to ensure fast token retrieval, and at reducing the tokens to be updated for enforcing new access requests to reduce the costs caused by update operations on the blockchain. We also proposed additional considerations that may be employed for further reducing the number of tokens necessary to enforce access restrictions.

References

1. Atallah, M., Blanton, M., Fazio, N., Frikken, K.: Dynamic and efficient key management for access hierarchies. ACM TISSEC **12**(3), 18:1–18:43 (January 2009)
2. Bacis, E., De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Rosa, M., Samarati, P.: Mix&Slice: Efficient access revocation in the cloud. In: Proc. of ACM CCS 2016. Vienna, Austria (October 2016)
3. Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A., Song, D.: Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution. arXiv preprint arXiv:1804.05141 (2018)
4. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Encryption policies for regulating access to outsourced data. ACM TODS **35**(2), 12:1–12:46 (April 2010)
5. De Capitani di Vimercati, S., Foresti, S., Livraga, G., Samarati, P.: Practical techniques building on encryption for protecting and managing data in the cloud. In: Ryan, P., Naccache, D., Quisquater, J.J. (eds.) The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday. Springer (2016)

6. De Capitani di Vimercati, S., Foresti, S., Livraga, G., Samarati, P.: Empowering owners with control in digital data markets. In: Proc. of IEEE CLOUD 2019. Milan, Italy (July 2019)
7. Di Francesco Maesa, D., Mori, P., Ricci, L.: Blockchain based access control. In: Proc. of DAIS. Neuchâtel, Switzerland (June 2017)
8. Foresti, S., Livraga, G.: Selective owner-side encryption in digital data markets: Strategies for key derivation. In: Proc. of SECRYPT 2021 (July 2021)
9. Kokoris-Kogias, E., Alp, E.C., Gasser, L., Jovanovic, P., Syta, E., Ford, B.: CALYPSO: Private data management for decentralized ledgers. PVLDB **14**(4), 586–599 (Dec 2020)
10. Liang, J., Li, S., Jiang, W., Cao, B., He, C.: OmniLytics: A blockchain-based secure data market for decentralized machine learning. In: Proc. of FL-ICML 2021 (July 2021)
11. Liu, K., Qiu, X., Chen, W., Chen, X., Zheng, Z.: Optimal pricing mechanism for data market in blockchain-enhanced Internet of Things. IEEE IoT-J **6**(6), 9748–9761 (2019)
12. Nguyen, D.C., Pathirana, P.N., Ding, M., Seneviratne, A.: Blockchain for secure EHRs sharing of mobile cloud based E-Health systems. IEEE Access **7**, 66792–66806 (2019)
13. Nguyen, L.D., Leyva-Mayorga, I., Lewis, A.N., Popovski, P.: Modeling and analysis of data trading on blockchain-based market in iot networks. IEEE IoT-J **8**(8), 6487–6497 (2021)
14. Shafagh, H., Burkhalter, L., Hithnawi, A., Duquennoy, S.: Towards blockchain-based auditable storage and sharing of IoT data. In: Proc. of CCSW. Dallas, TX, USA (November 2017)
15. Zhou, Y., Chen, J., He, B., Lv, L.: Data trading for blockchain-based data market in cyber-physical-social smart systems. In: Proc. of IEEE PIMRC 2021 (September 2021)
16. Zichichi, M., Ferretti, S., D'Angelo, G.: A framework based on distributed ledger technologies for data management and services in intelligent transportation systems. IEEE Access **8**, 100384–100402 (2020)
17. Zyskind, G., Nathan, O., Pentland, A.: Decentralizing privacy: Using blockchain to protect personal data. In: Proc. of IEEE SPW. San Jose, CA, USA (May 2015)