

Selective Owner-side Encryption in Digital Data Markets: Strategies for Key Derivation

Sara Foresti^a and Giovanni Livraga^b

Computer Science Department, Università Degli Studi di Milano, Via Celoria 18, Milano, Italy

Keywords: Digital Data Market, Selective Encryption, Key Derivation.

Abstract: The combined adoption of selective encryption and smart contracts deployed on blockchains allows data owners to maintain control over their data when traded on digital data market platforms. Selective encryption, combined with key derivation techniques, guarantees that only customers 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 parties, the possible economic incentives to be paid to the owners, and the exchange of the information necessary for resource decryption (i.e., updates to the key derivation structure) upon payment. However, operations on blockchains have a cost. In this paper, we propose two approaches for updating the key derivation structure to enable customers to access resources, while limiting access times to resources and the cost of write operations on the blockchain to enforce purchases.


1 INTRODUCTION


A great deal of interest is paid nowadays towards the development of spaces and platforms where data can be easily shared among interested subjects. Such spaces, typically called *digital data markets*, represent virtual places where *data owners*, acting as *data producers*, offer datasets, and *customers*, acting as *data consumers*, can access (parts of) these datasets. 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 hamper the diffusion of data-driven innovation is the (perceived) lack of control suffered by owners when they resort to these platforms. Completely delegating to a third party (such as the market provider) the storage and management of data, while relieving overhead at the owners' side, inevitably requires complete trust in the market provider, as it should be trusted to *i*) access all data; *ii*) maintain them confidential and not disclose them improperly to others; *iii*) correctly route the payment to the owner, if a price is to be paid to the owner for sharing a dataset with a customer. However, market providers are third parties that can more

realistically (in line with traditional data outsourcing scenarios) be considered *honest-but-curious*, meaning trusted for correctly managing data but not trusted for accessing their content. Hence, ensuring proper protection to the data shared in digital data markets, and maintaining owners in control over the customers with which data are shared, are key requirements for enabling a wide adoption of digital data markets.

To address these issues, a recent proposal combines *selective owner-side encryption* and *blockchain* (De Capitani di Vimercati et al., 2019). With selective owner-side encryption, data are stored in encrypted form. Encryption uses different keys, which are then distributed to the interested customers according to possible restrictions set by the owners. 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) cannot access the data content. When an interested customer requests access to a certain data collection of a given owner, their interaction and the possible economic transaction are managed via smart contracts deployed and executed on a blockchain. In this way the request, the payment, and the willingness of the owner to grant access to such data collection to that customer remains logged in the tamper-proof ledger of the blockchain, ensuring transparency and accountability.

^a  <https://orcid.org/0000-0002-1658-6734>

^b  <https://orcid.org/0000-0003-2661-8573>

The combined adoption of blockchain and selective owner-side encryption should however be carefully regulated. Selective owner-side encryption requires in fact the definition of *encryption keys* and of *tokens* enabling the *derivation* (i.e., computation) of one 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. At every purchase request by a customer, the key derivation structure (and hence the token catalog) needs to be updated to reflect and enforce the new granted access. However, writing on a blockchain inevitably implies an economic cost, and hence the updates to the key derivation structure should be carefully managed. In this paper, we investigate this issue and, after a brief discussion of basic concepts (Section 2), we investigate strategies (Section 3) that aim at minimizing the overall number of tokens in the system (Section 3.1) on the one hand, and which aim at minimizing the number of additional tokens needed to grant an access (Section 3.2), along with further optimizations (Section 3.3), on the other. We then discuss related works (Section 4) and present our conclusions (Section 5).

2 PRELIMINARIES

We consider a data market scenario, characterized by different data owners $O=\{o_1, \dots, o_l\}$ who are willing to sell their data (represented as a set $\mathcal{R}=\{r_1, \dots, r_m\}$ of resources) to interested customers $C=\{c_1, \dots, c_n\}$ through a platform made available by a data market provider.

To enable data owners to regulate access to resources by customers who paid for them, without the need of relying on the (curious) market provider, we adopt *selective encryption* as proposed in (De Capitani di Vimercati et al., 2019). Intuitively, selective encryption consists in encrypting (at the owner-side) different resources with different encryption keys, and in distributing encryption keys to customers in such a way that each customer can decrypt all and only the resources she is authorized to access. To limit the number of keys that customers need to manage, while preventing resource replication, selective encryption is typically combined with key derivation approaches. Key derivation enables the computation of the value of an encryption key k_j , leveraging the knowledge of another encryption key k_i through a publicly available token $t_{i,j}=k_j \oplus h(k_i, l_j)$, where h is a deterministic non-invertible encryption function, \oplus is the bitwise xor operator, and l_j is a public label associated with k_j (Atallah et al., 2009). Graphically, key derivation

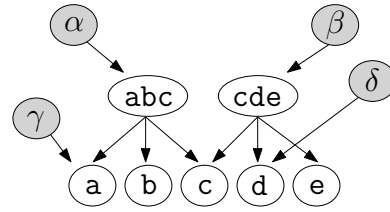


Figure 1: An example of key derivation hierarchy.

structures can be represented as graphs having a node for each encryption key and an edge from k_i to k_j if there is a token $t_{i,j}$ enabling the derivation of k_j starting from k_i . Any path in the graph represents a (direct or indirect) derivation relationship among encryption keys. In the remainder of this paper, for simplicity, we will denote with i or with k_i interchangeably the node representing key k_i .

To enforce access restrictions reflecting purchases (i.e., each customer can access in plaintext all and only the resources she purchased), each data owner o selectively encrypts her resources before storing them in the data market and distributes keys to customers according to purchases. The solution proposed in (De Capitani di Vimercati et al., 2019) is based on the definition of a key derivation structure with at least a key k_c for each customer c , a key k_r for each resource r , and a set of tokens enabling each customer to derive the keys of the resources that she purchased. Specifically, the structure is assumed to have a key for each set representing the *capability list* $\text{cap}(c)$ of a customer c (i.e., the set of resources that the customer purchased). The node of each customer c is connected, in the hierarchy, to the node representing her capability list $\text{cap}(c)$. The node of each capability list $\text{cap}(c)$ is connected (through a token) to other nodes in the key derivation structure in such a way that it is possible to (directly or indirectly) reach all and only the nodes representing the resources in the capability list (i.e., for each $r \in \text{cap}(c)$ there is a path from $\text{cap}(c)$ to r in the structure). This guarantees that each customer can derive the keys of all and only the resources in her capability list. To illustrate, consider a market storing five resources $\mathcal{R}=\{a, \dots, e\}$ owned by a single data owner, and four customers $C=\{\alpha, \beta, \gamma, \delta\}$ who purchased access to $\{a, b, c\}$, $\{c, d, e\}$, $\{a\}$, and $\{d\}$ respectively (i.e., $\text{cap}(\alpha) = \{a, b, c\}$, $\text{cap}(\beta) = \{c, d, e\}$, $\text{cap}(\gamma) = \{a\}$, and $\text{cap}(\delta) = \{d\}$). Figure 1 illustrates a key derivation hierarchy enabling customers to decrypt all and only the resources they purchased. For instance, from her key k_α , customer α can derive k_{abc} with token $t_{\alpha, abc}$ (edge (α, abc) in the figure). From key k_{abc} , she can then derive keys k_a , k_b , k_c and use these keys to decrypt resources a , b , and c , respectively. In the figure, we denote nodes representing

customers' keys with a gray background, to distinguish them from the nodes representing resources (or sets thereof), and omit commas and brackets (i.e., abc stands for $\{a, b, c\}$).

3 MINIMIZATION STRATEGIES

Data market scenarios are characterized by data owners joining the market to publish their resources, and by customers purchasing resources. The accommodation of the changes in the set of resources and access privileges requires the adaptation of the key derivation structure, and hence of the corresponding token catalog. Clearly, re-building the key derivation hierarchy from scratch at every purchase would be too expensive for the data owner, and would imply excessively long waiting times for customers. We therefore propose to update the hierarchy to accommodate purchases. Such updates should however be handled with care, for two main reasons: *i*) a large token catalog implies a higher response time for customers when accessing resources (due to the search for the tokens necessary for the access), and *ii*) write operations on the blockchain imply a cost and should then be performed with care. In this section, we propose two solutions aimed at limiting the impact of purchases on the token catalog. Section 3.1 illustrates an approach to minimize the overall number of token in the catalog. Section 3.2 focuses on minimizing the number of additional tokens for managing purchases (i.e., write operations on the blockchain). Section 3.3 presents some considerations to further reduce the number of tokens necessary for purchase management.

3.1 Minimize the Overall Number of Tokens

To limit response time for customers when accessing resources they have purchased, the size of the token catalog should be kept limited. Every access request implies a sequence of search operations in the catalog, for retrieving the tokens along the path connecting the node representing the requesting customer c with the node representing the resource r of interest. The size of the catalog has then a clear impact on response times. The problem of computing a key derivation hierarchy that minimizes the number of tokens is NP-hard (De Capitani di Vimercati et al., 2010). We therefore propose an heuristic approach aimed at locally modifying the key derivation hierarchy to accommodate the purchase, while keeping the number of tokens in the catalog low.

Let $\mathcal{H}(\mathcal{K}, \mathcal{T})$ be the key derivation hierarchy, with

\mathcal{K} the set of keys and \mathcal{T} the set of tokens. The hierarchy needs to be updated when: *i*) a new customer c joins the data market, *ii*) a new resource r is published in the market, and *iii*) a customer c purchases a resource r . The insertion of a customer implies only the creation of a new node c , whose key is agreed between the owner and the customer. Such a key is however not connected to the rest of the structure since her capability list is initially empty. Similarly, the publication of a new resource only implies the creation of a new node r , whose key is used to encrypt the resource, but which is not connected to the derivation structure by any token, since it does not belong to any capability list. The purchase of a resource r by a customer c requires instead a restructuring of the hierarchy, to guarantee the existence of a path from c to r .

Let us consider, for generality, the purchase of a set R of resources by a customer c . To keep the number of tokens in the system low, our solution is based on the idea of leveraging existing nodes and tokens to enable the derivation of the key used to encrypt each resource $r \in R$ starting from the key of customer c . Let k_{old} be the node in the hierarchy representing the capability list of the customer before the purchase (i.e., $cap(c)$), and k_{new} be the node representing the capability list of the customer after the purchase (i.e., $cap(c) \cup R$). To enforce the purchase of R by c , it is necessary to substitute the token enabling c to derive k_{old} from k_c with a new token that permits her to derive k_{new} from k_c . For instance, with reference to the example in Figure 1, if γ purchases resources b and c , the token enabling the derivation of k_a from k_γ needs to be substituted with a token enabling the derivation of k_{abc} from k_γ , as illustrated in Figure 2(a).

We note however that k_{new} might not belong to the key derivation structure and, in this case, it needs to be created and properly connected to the structure. In other words, it is necessary to insert into the hierarchy node k_{new} and a set of tokens enabling to derive k_r from k_{new} for each $r \in cap(c) \cup R$. To this aim, we connect k_{new} to a set K of nodes in hierarchy that represent subsets of $cap(c) \cup R$ and that guarantee complete coverage of such a set of resources (i.e., $\forall k_X \in K, X \subseteq cap(c) \cup R$ and $\bigcup_{k_X \in K} X = cap(c) \cup R$). Clearly, we aim at keeping the number of nodes in K low, to minimize the number of tokens in the system. The minimum coverage problem is NP-hard and we therefore propose a heuristic approach aimed at identifying a coverage for $cap(c) \cup R$ while limiting the number of nodes necessary for the coverage. To this purpose, we populate K starting from the larger subsets of $cap(c) \cup R$ in the hierarchy, until all the resources in $cap(c) \cup R$ are covered by at least a node in K . For instance, with reference to the example in Fig-

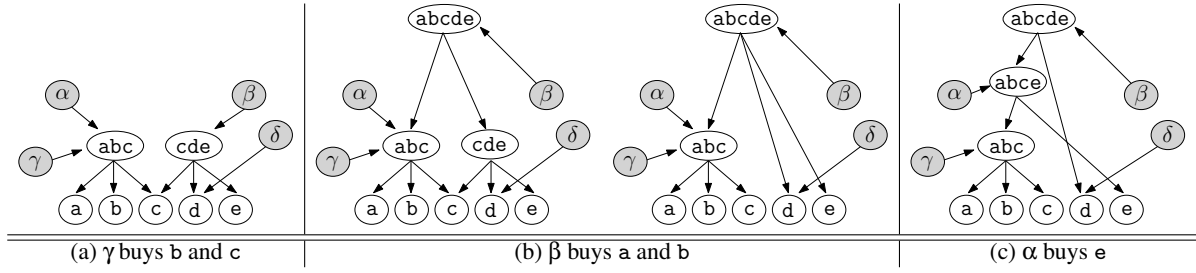


Figure 2: Evolution of the key derivation hierarchy in Figure 1 to enforce a sequence of purchases minimizing the overall number of tokens.

```

Purchase_Min_Token_Catalog( $c, R$ )
1: let  $Desc$  be the descendants of  $k_{cap(c)}$  and  $Anc$  be the ancestors of  $k_{cap(c)}$ 
2: if  $\exists c' \neq c \in C$  s.t.  $cap(c) = cap(c')$  AND  $|Desc| + |Asc| > |Desc'| + |Asc'|$ 
3: then remove  $k_{cap(c)}$  and all its incident edges from  $\mathcal{H}$ 
4: connect each node in  $Anc$  with each node in  $Desc$  via a set of tokens
5: else remove the token from  $k_c$  to  $k_{cap(c)}$ 
6: if  $k_{cap(c) \cup R} \notin \mathcal{K}$ 
7: then generate  $k_{cap(c) \cup R}$  and insert it into  $\mathcal{K}$ 
8: let  $K = \{k_X \in \mathcal{K} : X \subseteq cap(c) \cup R\}$ 
9:  $to\_cover := cap(c) \cup R$ 
10: while  $to\_cover \neq \emptyset$  do
11:   let  $k_X$  be the largest set of resources in  $K$  s.t.  $X \cap to\_cover \neq \emptyset$ 
12:    $K := K \setminus \{k_X\}$ 
13:    $to\_cover := to\_cover \setminus X$ 
14:   connect  $k_{cap(c) \cup R}$  to  $k_X$  via a token
15: connect  $k_c$  to  $k_{cap(c) \cup R}$  via a token
16: for each  $k_X \in \mathcal{K}$  s.t.  $cap(c) \cup R \subset X$  do
17:   let  $T$  be the tokens that are redundant if  $k_X$  is connected to  $k_{cap(c) \cup R}$ 
18:   if  $|T| > 2$ 
19:   then remove  $T$  from  $\mathcal{T}$  and connect  $k_X$  to  $k_{cap(c) \cup R}$  via a token
    
```

Figure 3: Management of resource purchase aimed to minimize the overall number of tokens.

ure 2(a), let us assume that β purchases a and b. It is first necessary to insert into the hierarchy a node representing $cap(\beta) = \{a, b, c, d, e\}$. According to our heuristics, such a node is then connected through a token to nodes abc and cde , as illustrated by the structure on the left hand side in Figure 2(b).

To reduce the number of tokens in \mathcal{T} , we note that node k_{old} can be possibly removed from the key derivation structure, if: *i*) no other customer shares the same capability list with c ; and *ii*) the removal of k_{old} reduces the number of tokens in \mathcal{T} . Indeed, the removal of k_{old} requires to directly connect all its ancestors with its all its descendants. For instance, considering the purchase of a and b by β , node cde can be removed from the key derivation structure (see the structure on the right hand side of Figure 2(b)), saving one token with respect to the structure on the left hand side in Figure 2(b). The insertion of node k_{new} can also be leveraged to further reduce the number of tokens in the hierarchy. To this purpose, it is possible to analyze each node k_X in the hierarchy representing a superset of $cap(c) \cup R$. If such a node can be connected to k_{new} removing its connection to at least two of its descendants, as they become redundant, we enforce such a change. For instance, considering the

hierarchy in Figure 2(b), assume that customer α purchases resource e. The new node representing $cap(\alpha)$ can become direct descendant of node $abcde$, saving one token in the structure (see Figure 2(c)).

Figure 3 illustrates the pseudo-code of the procedure enforcing the purchase of a set R of resources by a customer c , aimed at minimizing the overall number of tokens in the catalog. Note that the procedure can manage the purchase of a single resource r , by simply setting R to the singleton set $\{r\}$ of resources.

3.2 Minimize the Number of Additional Tokens

The strategy illustrated in Section 3.1, while effective in limiting the cost for customers in accessing data, implies an economic cost for the data owner proportional to the number of tokens inserted and deleted from the catalog. Indeed, both insertion and deletion of tokens imply a write operation on the catalog, and hence on the blockchain. In this section, we propose an alternative approach for managing purchases, aimed at minimizing the operations on the blockchain. To this aim, we propose to prevent token deletion (the removal of tokens is not necessary for the enforcement of purchases) and minimize the number of additional tokens.

Let us consider the purchase of a set R of resources by a customer c . The capability list of c needs to be updated to include also the resources in R , that is $cap(c) \cup R$. Therefore, we need to connect the node of c with a node in the hierarchy representing the set $cap(c) \cup R$ of resources. If such a node already belongs to the hierarchy, we can simply add a token enabling the derivation of $k_{cap(c) \cup R}$ from k_c , as showed by the structure in Figure 4(a) illustrating the structure in Figure 1 updated to accommodate the purchase of b and c by γ : since the structure already contains a node for the new capability list abc of γ , it is sufficient to create a token enabling the derivation of k_{abc} from k_γ (note that in the figure, for the sake of readability, we did not report the old token connecting γ to a). On the

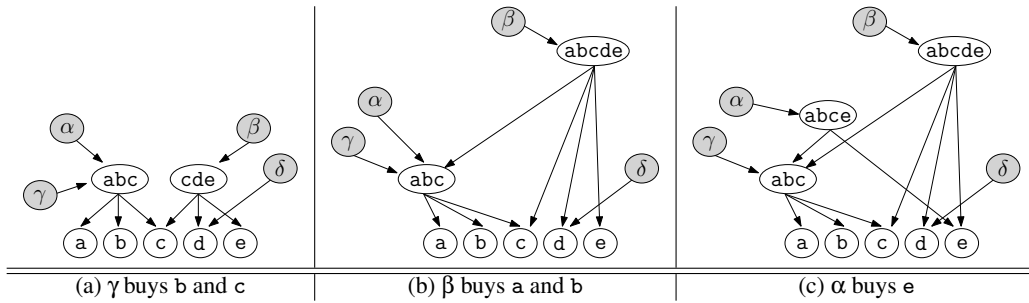


Figure 4: Evolution of the key derivation hierarchy in Figure1 to enforce a sequence of purchases minimizing the number of additional tokens.

contrary, if such a node does not exist, we envision the following two possible scenarios.

- **New node:** insert a node in the hierarchy representing $\text{cap}(c) \cup R$ and connect it to the node representing $\text{cap}(c)$ (and hence indirectly to the corresponding resources) and, directly or indirectly, to the nodes of the resources in R .
- **Rename node:** connect (directly or indirectly) the node representing $\text{cap}(c)$ to the nodes of the resources in R (i.e., node representing $\text{cap}(c)$ is relabeled to represent $\text{cap}(c) \cup R$). This strategy saves the insertion and deletion of one token (the one from k_c to $k_{\text{cap}(c) \cup R}$, as it already exists) compared to the previous strategy, but it is not always viable. Specifically, this approach can be adopted only if the following two conditions hold: *i*) no other customer c' shares the (old) capability with c , since otherwise c' too would be granted access to R without being authorized; and *ii*) all the ancestors of the node represent a superset of $\text{cap}(c) \cup R$, since otherwise this could enable customers who can derive the ancestors of the node to access R without paying for such resources.

For instance, consider the purchase of resources a and b by β illustrated in Figure 2(b), starting from the hierarchy in Figure 2(a). This purchase can be managed by renaming node cde into $abcde$ (see Figure 4(b)) paying the insertion of 2 tokens, in contrast to the insertion of 4 tokens and the removal of 4 tokens paid by the solution in Figure 2(b). On the contrary, the purchase by α of e cannot be managed with a simple renaming of node abc , since γ would gain access to e without paying for such a resource. The structure is then updated inserting a new node, representing $abce$, connected to abc and to e (see Figure 4(c)).

Independently from the strategy chosen for having a node representing $\text{cap}(c) \cup R$ in the hierarchy, such a node (which is already connected to the resources in $\text{cap}(c)$) needs to be connected to the resources in R . A straightforward solution consists in adding a token enabling the derivation of the key k_r used to en-

Purchase_Min_Additional_Tokens(c, R)

```

1: if  $k_{\text{cap}(c) \cup R} \notin \mathcal{X}$ 
2:   let  $Anc$  be the ancestors of  $k_{\text{cap}(c)}$ 
3:   if  $\nexists c' \neq c \in C$  s.t.  $\text{cap}(c') = \text{cap}(c)$  AND  $\forall k_X \in Anc, \text{cap}(c) \cup R \subset X$ 
4:     then relabel  $k_{\text{cap}(c)}$  as  $k_{\text{cap}(c) \cup R}$ 
5:   else generate  $k_{\text{cap}(c) \cup R}$  and insert it into  $\mathcal{X}$ 
6:     connect  $k_{\text{cap}(c) \cup R}$  to  $k_{\text{cap}(c)}$  via a token
7:   let  $K = \{k_X \in \mathcal{X} : X \subseteq \text{cap}(c) \cup R\}$ 
8:    $to\_cover := R$ 
9:   while  $to\_cover \neq \emptyset$  do
10:    let  $k_X$  be the largest set of resources in  $K$  s.t.  $X \cap to\_cover \neq \emptyset$ 
11:     $K := K \setminus \{k_X\}$ 
12:     $to\_cover := to\_cover \setminus X$ 
13:    connect  $k_{\text{cap}(c) \cup R}$  to  $k_X$  via a token
14: connect  $k_c$  to  $k_{\text{cap}(c) \cup R}$  via a token

```

Figure 5: Management of resource purchase aimed to minimize the additional number of tokens.

crypt each resource $r \in R$ from $k_{\text{cap}(c) \cup R}$. This simple solution implies the creation of $|R|$ new tokens. We note however that, if the hierarchy already includes a node representing R , node $\text{cap}(c) \cup R$ can be directly connected to it, enabling the derivation of k_r , $\forall r \in R$, paying one token only. In general, if the hierarchy includes a set K composed of less than $|R|$ nodes representing subsets of $\text{cap}(c) \cup R$ (and hence also of R) that completely cover R , the number of additional tokens can be reduced to $|K|$ by connecting $\text{cap}(c) \cup R$ with each of the nodes in K . Clearly, we aim at minimizing the number of nodes in K . We then follow a strategy similar to the one discussed in Section 3.1, and first consider nodes covering larger subsets of R as candidates for insertion into K . For instance, with reference to our running example, node $abcde$ is connected to node abc , which completely covers the set $R = \{a, b\}$ of resources purchased by β (see Figure 4(b)). Indeed, connecting $abcde$ to each purchased resource would cost one additional token.

Figure 5 illustrates the pseudo-code of the function enforcing the purchase of a set R of resources by a customer c , aimed at minimizing the number of tokens inserted into the catalog for the purchase.

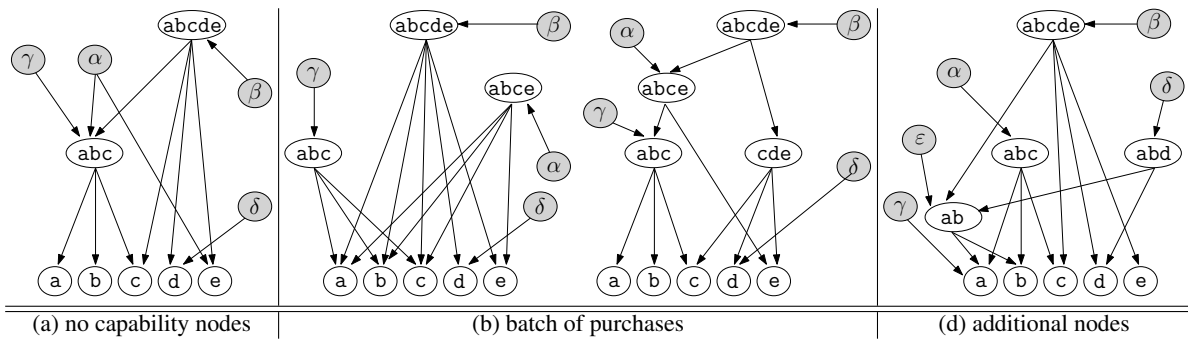


Figure 6: Evolution of the key derivation hierarchy, considering the optimizations.

3.3 Discussion

The strategy illustrated in Section 3.2 can take advantage of additional considerations that can possibly further reduce the number of tokens inserted into the catalog to enforce purchases. In this section, we discuss further optimizations that could be adopted.

- Remove the assumption of having a node representing the capability list of each customer.** Having a node representing the capability list of each customer implies the rename or the creation of a new node at each purchase of a resource (or set thereof). Removing this assumption could reduce the number of tokens that need to be inserted into the catalog to manage a purchase operation. Consider, as an example, a customer c with capability list $\text{cap}(c)$, and assume that c purchases access to a set R of resources and that the key derivation hierarchy already includes a key k_R . If we do not require the presence of a node representing the set $\text{cap}(c) \cup R$ of resources, it is sufficient to create a token enabling the derivation of k_R from k_c . On the contrary, the need for node $\text{cap}(c) \cup R$ implies the insertion of at least two tokens in the hierarchy: one enabling the derivation of $k_{\text{cap}(c) \cup R}$ from k_c , and another one enabling the derivation of k_R from $k_{\text{cap}(c) \cup R}$. Also, if $\text{cap}(c) \cup R$ is not represented by a node in the hierarchy, it is also necessary to insert a token from $k_{\text{cap}(c) \cup R}$ to $k_{\text{cap}(c)}$ (i.e., three tokens). For instance, with reference to our running example, Figure 6(a) illustrates the key derivation hierarchy obtained managing the purchases of our running example, according to this optimization strategy. This solution saves the insertion of 3 tokens for the management of the purchase of e by α . Note that we did not manage the purchase of a and b by β and of b and c by γ connecting directly the customer's node to the resources' nodes since this would not provide any cost reduction.

- Manage purchases in batch.** The solutions illustrated in this section implicitly assume that purchases are managed when they are submitted to the data owner, who is expected to accommodate each purchase independently from the others. However, the management of purchases in batches could provide considerable advantages for the data owner. As a first advantage, the data owner would not need to be always available to manage resource purchases, but decide (in agreement with customers) to enforce changes to the key derivation hierarchy periodically (e.g., once a day). As a second advantage, the management of multiple purchases, by the same or by multiple customers over one or more resources, could provide advantages also in terms of the management of the key derivation hierarchy, enabling higher economic savings for the write operations in the blockchain. Intuitively, the data owner can operate all the optimizations on the sub-hierarchy necessary for managing the purchases in the batch and possibly reduce the number of tokens with respect to the management of each purchase singularly taken. Clearly, there is an advantage in managing together a batch of purchases by multiple customers if they refer to the same resource (or set thereof). Consider, as an example, three customers c_x , c_y , and c_z who all purchase resources r_a and r_b . In this case, it would be convenient for the data owner to create a node representing the set $\{r_a, r_b\}$ of resources and connect c_x , c_y , and c_z to such a node, which is in turn connected to r_a and r_b . This strategy implies the insertion of 5 tokens in the key derivation hierarchy. If the three purchases were managed in sequence, at their submission time, the cost would be of 6 tokens (2 for each customer for connecting their node to the node of each of the 2 resources). Intuitively, the data owner could locally perform the optimizations illustrated in Section 3.1 on the portion of the hierarchy affected by the batch of purchases

to be managed, and then connect it to the public key derivation structure when the number of additional tokens has been reduced as much as possible. For instance, with reference to our running example, Figure 6(b) illustrates on the left the key derivation hierarchy obtained managing, in the order, the purchase of a by β , of c by γ , of b by β , of e by α , and of b by γ starting from the initial configuration in Figure 1. Figure 6(b) illustrates on the right the key derivation hierarchy obtained managing the same purchases in batch. While the first strategy implies the addition of 8 tokens, the second one costs only 6 additional tokens.

- *Additional nodes.* The data owner might consider, when managing purchases, to insert additional nodes in the hierarchy to possibly reduce the number of tokens that will be needed in the future. Indeed, the presence of a node in the hierarchy representing a set R of resources could be profitably used by any customer buying a superset of R , saving in the number of tokens. Indeed, if at least two customers buy a superset of R , there is an advantage in having R in the hierarchy, in terms of the number of tokens paid by the data owner to manage the two purchases by the two customers. The data owner can therefore decide, when a customer buys a set R of resources, to materialize the node representing R , if she is confident on the fact that other customers will be interested in the same set. The data owner pays an additional token when first inserting R , but she may experience a saving in the future. For instance, with reference to the initial configuration of the key derivation hierarchy in Figure 1, to manage the purchase of resources a and b by β , the data owner might decide to create a new node ab paying 3 tokens. If, after some time, also δ and ϵ buy these resources, it would be sufficient to insert a token from k_{abd} to k_{ab} for δ and from k_ϵ to k_{ab} for ϵ , saving on the number of tokens inserted to manage the three purchases (7 tokens instead of 9 tokens). Figure 6(c) illustrates the resulting key derivation hierarchy.

4 RELATED WORK

The adoption of selective encryption for enforcing access restrictions in digital data market scenarios, coupled with smart contracts deployed on a blockchain, has first been proposed in (De Capitani di Vimercati et al., 2019). Our solution, while sharing with the proposal in (De Capitani di Vimercati et al., 2019) the use of selective encryption and key derivation for en-

abling data owners to maintain control over their resources in the data market, nicely complements it. Indeed, our techniques for minimizing the size of the token catalog and for limiting the number of additional tokens implied by the management of each purchase can be used in combination with the protocols for resource purchase presented in (De Capitani di Vimercati et al., 2019).

The adoption of selective encryption, possibly combined with key derivation, has been widely adopted in data outsourcing scenarios, which are characterized by data owners storing their resources on the premises of non fully trusted cloud providers (e.g., (Bacis et al., 2016; De Capitani di Vimercati et al., 2010; De Capitani di Vimercati et al., 2016)). These approaches however operate in a different scenario and aim at enforcing a (quite static) authorization policy defined by the data owner. Also, 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.

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., (Di Francesco Maesa et al., 2017; Kokoris-Kogias et al., 2020; Nguyen et al., 2019; Nguyen et al., 2021; Shafagh et al., 2017; Zichichi et al., 2020; Zyskind et al., 2015)), and privacy and security in cloud computing (e.g., (Donida Labati et al., 2020; Zhang et al., 2020)). These approaches are however complementary to ours, as they do not consider selective encryption and key derivation for enforcing access restrictions to traded resources, and do not consider the peculiarities of data markets.

5 CONCLUSIONS

We studied the management of purchases of resources in a data market scenario, by properly modifying the key derivation structure used to enforce access restrictions. We proposed two alternative solutions, aimed at minimizing the overall number of tokens and the number of additional tokens necessary to support each purchase, respectively. We also discussed improvements for further reducing the number of tokens necessary to enforce purchases. Our work leaves room to future works, aimed at comparing the two strategies in a real world environment to determine which solution should be preferred in different application scenarios.

ACKNOWLEDGMENTS

This work was supported in part by the EC within the H2020 Program under projects MOSAICrOWN and MARSAL.

REFERENCES

- Atallah, M., Blanton, M., Fazio, N., and Frikken, K. (2009). Dynamic and efficient key management for access hierarchies. *ACM TISSEC*, 12(3):18:1–18:43.
- Bacis, E., De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Rosa, M., and Samarati, P. (2016). Mix&Slice: Efficient access revocation in the cloud. In *Proc. of ACM CCS*, Vienna, Austria.
- De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P. (2010). Encryption policies for regulating access to outsourced data. *ACM TODS*, 35(2):12:1–12:46.
- De Capitani di Vimercati, S., Foresti, S., Livraga, G., and Samarati, P. (2016). Practical techniques building on encryption for protecting and managing data in the cloud. In Ryan, P., Naccache, D., and Quisquater, J.-J., editors, *The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*. Springer-Verlag Berlin Heidelberg.
- De Capitani di Vimercati, S., Foresti, S., Livraga, G., and Samarati, P. (2019). Empowering owners with control in digital data markets. In *Proc. of IEEE CLOUD*, Milan, Italy.
- Di Francesco Maesa, D., Mori, P., and Ricci, L. (2017). Blockchain based access control. In *Proc. of DAIS*, Neuchâtel, Switzerland.
- Donida Labati, R., Genovese, A., Piuri, V., Scotti, F., and Vishwakarma, S. (2020). Computational intelligence in cloud computing. In Kovács, L., Haidegger, T., and Szakál, A., editors, *Recent Advances in Intelligent Engineering: Volume Dedicated to Imre J. Rudas' Seventieth Birthday*. Springer International Publishing.
- Kokoris-Kogias, E., Alp, E. C., Gasser, L., Jovanovic, P., Syta, E., and Ford, B. (2020). CALYPSO: Private data management for decentralized ledgers. *PVLDB*, 14(4):586–599.
- Nguyen, D. C., Pathirana, P. N., Ding, M., and Seneviratne, A. (2019). Blockchain for secure EHRs sharing of mobile cloud based E-Health systems. *IEEE Access*, 7:66792–66806.
- Nguyen, L. D., Leyva-Mayorga, I., Lewis, A. N., and Popovski, P. (2021). Modeling and analysis of data trading on blockchain-based market in IoT networks. *IEEE IoT-J*, 8(8):6487–6497.
- Shafagh, H., Burkhalter, L., Hithnawi, A., and Duquennoy, S. (2017). Towards blockchain-based auditable storage and sharing of IoT data. In *Proc. of CCSW*, Dallas, TX, USA.
- Zhang, Y., Deng, R. H., Xu, S., Sun, J., Li, Q., and Zheng, D. (2020). Attribute-based encryption for cloud computing access control: A survey. *ACM CSUR*, 53(4).
- Zichichi, M., Ferretti, S., and D'Angelo, G. (2020). A framework based on distributed ledger technologies for data management and services in intelligent transportation systems. *IEEE Access*, 8:100384–100402.
- Zyskind, G., Nathan, O., et al. (2015). Decentralizing privacy: Using blockchain to protect personal data. In *Proc. of IEEE SPW*, San Jose, CA, USA.