Empowering Owners with Control in Digital Data Markets

Sabrina De Capitani di Vimercati, Sara Foresti, Giovanni Livraga, Pierangela Samarati
Computer Science Department, Università degli Studi di Milano
Milan, Italy
{*sabrina.decapitani, sara.foresti, giovanni.livraga, pierangela.samarati*}*@unimi.it*

***Abstract*—We propose an approach for allowing data owners to trade their data in digital data market scenarios, while keeping control over them. Our solution is based on a combination of selective encryption and smart contracts deployed on a blockchain, and ensures that only authorized users who paid an agreed amount can access a data item. We propose a safe interaction protocol for regulating the interplay between a data owner and subjects wishing to purchase (a subset of) her data, and an audit process for counteracting possible misbehaviors by any of the interacting parties. Our solution aims to make a step towards the realization of data market platforms where owners can benefit from trading their data while maintaining control.**

***Keywords*-Digital data market, Selective encryption, Key derivation, Blockchain, Smart contract**

I. Introduction

In our hyper-connected society, data are continuously generated at the astonishing pace of quintillions of bytes each day, and the trend is forecasted to grow in the near future also due to the incredible diffusion of the IoT technology. These data, often related to individuals, have a huge value for creating knowledge [1] (data is oftentimes referred to as the oil of the future), and this has fostered a vision towards the development of *digital data markets*, where data are monetized and traded. In the current scenario, the value is most of the times cashed by a few big players on the market, which have a revenue in collecting and trading data. The problem is that this is frequently done without the explicit consent of the individuals to whom the data refer, and this is increasingly seen as a violation of their fundamental rights. Recent laws and regulations, such as the EU General Data Protection Regulation (GDPR), clearly state that data subjects (i.e., users to whom data refer) should always be in control of their data. A natural corollary is the need for markets where data owners (be them data subjects/controllers or more generally users with the rights to share data) can directly contribute their data, and monetize them by making them available to others in a controlled way.

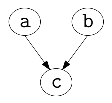
The realization of such data market platforms clearly raises a number of issues, for instance due to high requirements for storage, networking, and computation resources. Having powerful and elastic platforms that are already used today for storing and processing huge amounts of data, cloud service providers are in a unique position for unlocking the full potential of data markets. As a matter of fact, in many real-world scenarios cloud platforms are already operating as large data hubs. To fully realize such a data market vision and satisfy data protection requirements, individuals should have adequate guarantees that the data they make available in the data market are properly protected, meaning accessed only with their consent and with an adequate compensation. A naive solution could delegate the management of accesses and payments to the data market provider. This would however require trust in the provider by both the owners and the users purchasing data, which is a strong and unrealistic assumption.

In this paper, we propose a solution for empowering data owners to monetize their data, while keeping control over them, in data market platforms. We consider dynamic scenarios where, as time passes, *data owners* (users) can upload to the market platform new datasets, and *processors* (external users) can purchase access to them. We do not rely on a trusted market provider and, to properly protect data, we adopt owner-side encryption. The contribution of this paper is three-fold. First, we propose a data protection mechanism for data markets based on selective encryption, where each data item is encrypted with a different key, to enforce fine-grained access restrictions. The burden of key management is limited through the adoption of a key derivation strategy. Second, we present a safe interaction protocol for regulating payments for resources based on blockchain and smart contracts. Third, we propose an audit process that, leveraging the combined adoption of selective encryption, blockchain, and smart contracts, counteracts possible misbehaviors in case any of the interacting subjects acts maliciously.

The remainder of this paper is organized as follows. Section II gives some preliminaries and basic concepts. Section III characterizes our reference scenario and its requirements. Section IV illustrates our solution based on selective encryption for protecting data. Section V discusses our blockchain-based interaction protocol and our audit process for managing data purchases and for recognizing and discouraging misbehaviors. Section VI presents some considerations on our approach. Section VII discusses related works. Finally, Section VIII concludes the paper.

II. Preliminaries

Our solution combines three building blocks: *i)* key derivation; *ii)* blockchain; and *iii)* smart contracts.



$\mathcal{T}$

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

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

Key derivation. Key derivation permits to derive the value of an encryption key k_y from the knowledge of another encryption key k_x and of a public label l_y (i.e., a piece of information) associated with k_y [2], [3]. The derivation of k_y from k_x is enabled by a public token $t_{x,y}$ computed as $k_y \oplus h(k_x, l_y)$, with $\oplus$ the bitwise `xor` operator, and h a deterministic non-invertible cryptographic function. The derivation relationship between keys can be *direct*, via a single token, or *indirect*, through a chain of tokens. Key derivation structures can be graphically represented as directed acyclic graphs, where vertices represent ecryption keys (and their labels), and edges represent tokens. Tokens are physically stored in a public catalog $\mathcal{T}$. Figure 1 illustrates an example of derivation among three keys $k_{\mathtt{a}}$, $k_{\mathtt{b}}$, and $k_{\mathtt{c}}$, and the corresponding token catalog $\mathcal{T}$. For simplicity, in our examples we use x to denote the label of key k_x, and use the label of a key to denote the corresponding vertex (e.g., vertex a in Figure 1 represents key $k_{\mathtt{a}}$ and its label). In the following, when clear from the context, we will use the terms keys and vertices (tokens and edges, respectively) interchangeably.

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

Smart contracts. Smart contracts are a powerful tool for establishing contracts among multiple, possibly distrusting, parties. A smart contract is a software running on top of a blockchain and defines a set of rules, on which the interacting parties agree. It can be seen as a set of 'if-then' instructions, defining triggering conditions and subsequent actions capturing and formalizing the clauses of a contract to be signed by the parties. The execution of a smart contract can be trusted for correctness thanks to the underlying blockchain consensus protocols, meaning that all the conditions of the agreement modeled by the contract are certainly met and validated by the network. However, smart contracts and their execution lack confidentiality and privacy, as plain visibility over the content of a contract and over the data it manipulates is necessary for validation [4].

III. Scenario and Requirements

We address the problem of allowing subjects to leverage the availability of data market platforms for trading their data with interested processors (i.e., entities that need to perform some processing over them). Our scenario is then characterized by a set $\mathcal{O}$ of data owners on one side, and a set $\mathcal{P}$ of processors on the other side, who interact through the market platform to sell and buy data, modeled as a generic set $\mathcal{R}$ of resources. In this paper, we investigate two main issues: *i)* ensuring adequate data protection, and *ii)* ensuring that owners and processors can profitably leverage data market platforms for doing business. Our first goal aims at maintaining the owner in control of her data, by defining a solution that enables a processor to access a resource only upon payment, and which ensures that the owner is aware of which processor has access to which resource. Due to the involvement of money and since processors and data owners might not fully trust each other, our second goal aims at defining mechanisms for counteracting misbehaviors from owners and processors (e.g., a malicious owner does not provide access to a paid resource or, conversely, a processor claims her money back by maliciously declaring the owner did not grant access despite the payment). In designing our solution, we keep the following requirements in mind:

R1. the content of published resources must remain protected, and only authorized processors who purchased access to a resource can access its content;

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

R3. after a processor p purchases access to a resource, the owner cannot claim that payment has not been received (and refuse to grant access to p);

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

While the first two requirements deal with ensuring data protection, the latter two reduce the possibility of misbehaviors from both the data owner and the processors. We satisfy *R1* by protecting resources through selective owner-side encryption. We satisfy *R2* by monitoring access rights through blockchain and smart contracts. We satisfy *R3* and *R4* by counteracting misbehaviors through an audit process that incentivizes all parties to behave correctly.

Example 3.1: For the sake of concreteness, we refer our discussion to the data generated by a fitness tracker that measures a number of parameters (e.g., heart rate, steps and movements, sleep quality). The measurements are collected throughout the whole day, and the owner of the device (and hence also of the generated data) can download them for analysis. Since these data can be of interest to a multitude of subjects (e.g., researchers and pharmaceutical companies), 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 six sets of measurements $\mathcal{R}=\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e},\mathtt{f}\}$, each one consisting of the measurements collected in two months (i.e., one year of measurements). These data are of interest for four processors $\mathcal{P}=\{\mathtt{w},\mathtt{x},\mathtt{y},\mathtt{z}\}$, which over time buy access to resources.

IV. Protecting Resources

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

A. Authorization Policy and Key Derivation Structure

The authorization policy $\mathcal{A}$ regulating access to resources needs to reflect purchases of the processors. We represent the authorization policy by means of the *capability lists* of the processors in $\mathcal{P}$, where $\mathsf{cap}(p)$ represents the set of resources for which processor $p\in\mathcal{P}$ is authorized (i.e., for which the owner o has received a payment from p). Every time processor p buys access to a resource $r\in\mathcal{R}$, r will be added to her capability list (i.e., $\mathsf{cap}(p)=\mathsf{cap}(p)\cup\{r\}$).

To allow fine-grained access control as demanded by our scenario, without the need to trust the data market to enforce access privileges, we leverage *selective owner-side encryption* [3]. Intuitively, selective owner-side encryption consists in encrypting, at the owner-side, different resources with different keys, and in distributing keys to processors in such a way that each processor can decrypt all and only the resources she is authorized to access. Since the encryption layer is provided at the owner side, resources self-enforce the access restrictions defined over them and their content is protected also to the eyes of the market provider. The data owner can then make her resources available to the market platform (which can be hosted, for instance, on the cloud), with the guarantee that only processors knowing the encryption keys will be able to decrypt resources. A straightforward solution to enforce access restrictions through selective encryption consists in encrypting each resource with a different key, and in distributing to each processor the keys of the resources in her capability list. However, this practice would imply a considerable key management burden for processors. To mitigate such overhead, we adopt key derivation (see Section II). Intuitively, each processor p agrees a key k_p with the data owner, who publishes a token allowing p to compute k_r from k_p for each r in $\mathsf{cap}(p)$ (if a same processor purchases resources from different owners, she can create a set of tokens enabling her to compute the key shared with each owner from a single (secret) master key). While effective, this simple solution could not be efficient since it might create more tokens than necessary. To reduce the number of tokens, the key derivation structure is typically enriched with additional vertices whose key is used for derivation only [3], [5]. While in cloud-based scenarios such additional vertices are typically associated with groups of users, the considered scenario would benefit from the definition of keys associated with groups of resources. Indeed, such an approach guarantees that each resource r has a different encryption key (the one corresponding to singleton set $\{r\}$). Also, processors join the system purchasing subsets of resources and it is therefore natural to think in terms of groups of resources in contrast to groups of users. Formally, a key derivation structure is defined as follows.

Definition 4.1 (Key derivation structure): Given a set $\mathcal{R}=\{r_1,\ldots,r_n\}$ of resources and a set $\mathcal{P}=\{p_1,\ldots,p_m\}$ of processors, a *key derivation structure* over $\mathcal{R}$ and $\mathcal{P}$ is a directed acyclic graph $G(V,E)$ such that:

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

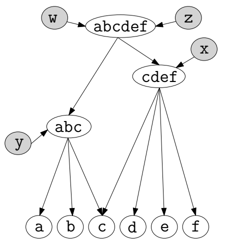
According to the definition above, vertices in the key derivation structure represent processors or sets of resources (Condition 1). Also, the derivation structure has a vertex for each processor and for each resource (Condition 2). Vertices representing processors have only outgoing edges ending at vertices representing (sets of) resources, while edges connecting sets of resources satisfy the subset containment relationship, that is, each vertex is connected to vertices representing subsets of its resources (Condition 3). Each processor p knows the key of its vertex v_p and each resource r is encrypted with the key of its vertex v_r. With reference to Example 3.1, Figure 2(b) illustrates an example of key derivation structure with a vertex for each processor (gray), a vertex for each resource, and additional vertices for subsets of resources. As already noted, for simplicity in the figures we denote each vertex v_x with x (e.g., $\mathtt{a}$ is the vertex for resource $\mathtt{a}$ and $\mathtt{x}$ is the vertex of processor $\mathtt{x}$.)

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

Definition 4.2 (Correctness): Given an authorization policy $\mathcal{A}$ over a set $\mathcal{R}$ of resources and a set $\mathcal{P}$ of processors,

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

(a)



(b)

Figure 2. An example of authorization policy (a), and of a key derivation structure enforcing it (b)

a key derivation structure $G(V,E)$ over $\mathcal{R}$ and $\mathcal{P}$ *correctly enforces* $\mathcal{A}$ iff:
$\forall r \in \mathcal{R}, \forall p \in \mathcal{P} : r \in \mathsf{cap}(p) \Leftrightarrow \exists$ a path in G from v_p to v_r.

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

Example 4.1: Consider the authorization policy in Figure 2(a) for the sets of resources and processors of Example 3.1. Figure 2(b) illustrates an example of a key derivation structure enforcing the policy. The structure includes one vertex for each resource, one vertex for each processor, and three vertices representing their capability lists. Edges of the structure connect processors to their capability lists, and sets of resources according to the subset containment relationship, in such a way to guarantee coverage (e.g., vertex abc is covered by a, b, and c). It is immediate to verify that the key derivation structure in Figure 2(b) correctly enforces the authorization policy in Figure 2(a), since it enables each processor to reach all and only the resources she is entitled to access.

B. Resources and Access Management

The key derivation structure is updated whenever new resources are published and/or purchased.

The publication of resource r is easily enforced by simply inserting in the structure a vertex for r. The owner generates an encryption key k_r and a label l_r for the vertex, encrypts r with k_r, and publishes the encrypted resource on the market. To illustrate, consider the structures in Figure 3. The leftmost one represents the structure for our running example after the publication of the six resources (since no authorization has been granted yet, no processor vertex is present).

The purchase of a set R of resources by processor p is enforced by procedure **Enforce_Purchase** (Figure 4). Note that, in the figure and in the following discussion, the generation of vertices (edges, resp.) implies the generation of their keys and labels (tokens, resp.). The procedure takes as input the requesting processor p, its current capability list $\mathsf{cap}(p)$, the set R of resources she wants to buy, and the key derivation structure $G(V,E)$. It updates the structure enabling p to derive the keys necessary to decrypt the resources in $\mathsf{cap}(p) \cup R$. The procedure fist checks whether the structure already contains a vertex v_p for p (i.e., if p already purchased resources in the market) and, if this is not the case, it creates vertex v_p and the corresponding key and label (lines 1–2). If the vertex $v_{\mathsf{cap}(p)}$ representing the (old) capability list of p does exist, the procedure deletes $(v_p, v_{\mathsf{cap}(p)})$ (lines 3–4). It then checks whether the removal of $v_{\mathsf{cap}(p)}$ could reduce the number of edges [3] and, if this is the case, it removes $v_{\mathsf{cap}(p)}$ connecting all its parents to all its children (lines 5–13). The procedure then updates the capability list $\mathsf{cap}(p)$ by including the resources in R (line 14). If the key derivation structure already includes vertex $v_{\mathsf{cap}(p)}$ representing the new capability list of p, then v_p is simply connected to $v_{\mathsf{cap}(p)}$, and the procedure terminates (line 15). Otherwise, $v_{\mathsf{cap}(p)}$ first needs to be created, and only at this point an edge is created to enable the derivation of $v_{\mathsf{cap}(p)}$ from v_p (lines 16–19). To guarantee the correctness of the key derivation structure, all the resources in $\mathsf{cap}(p)$ should be reachable from $v_{\mathsf{cap}(p)}$ (Definition 4.2). Hence, the procedure identifies the set *Desc* of vertices representing subsets of resources in $\mathsf{cap}(p)$, and selects a subset *Cover* of vertices in *Desc* forming a set cover for $\mathsf{cap}(p)$ (lines 20–21). Vertex $v_{\mathsf{cap}(p)}$ is connected to the vertices in *Cover* (line 22). The procedure finally checks if it is possible to further reduce the number of edges [3] thanks to the presence of $v_{\mathsf{cap}(p)}$. To this aim, the procedure identifies the set *Par* of the vertices representing supersets of $\mathsf{cap}(p)$, and the set of *DescCover* of the vertices reachable from a vertex in *Cover* (lines 23–24). Indeed, if a vertex v_{par} in *Par* is directly connected to more than one vertex in *Cover* and/or in *DescCover*, the insertion of $v_{\mathsf{cap}(p)}$ as an intermediate vertex and removal of edges from v_{par} to the vertices in *Cover* and/or in *DescCover* (lines 25–31) reduces the number of edges.

Example 4.2: Figure 3 illustrates the evolution of the leftmost structure to enforce the sequence of requests illustrated at the bottom of the figure. The first two requests insert vertices abc and cdef for w and x respectively. The third request does not insert vertices since $\mathsf{cap}(\mathtt{y})$=abc already belongs to the structure. The fourth request inserts bc for z. Note that connecting abc to bc saves an edge. The fifth request inserts a vertex for the entire set $\mathcal{R}$, for which z

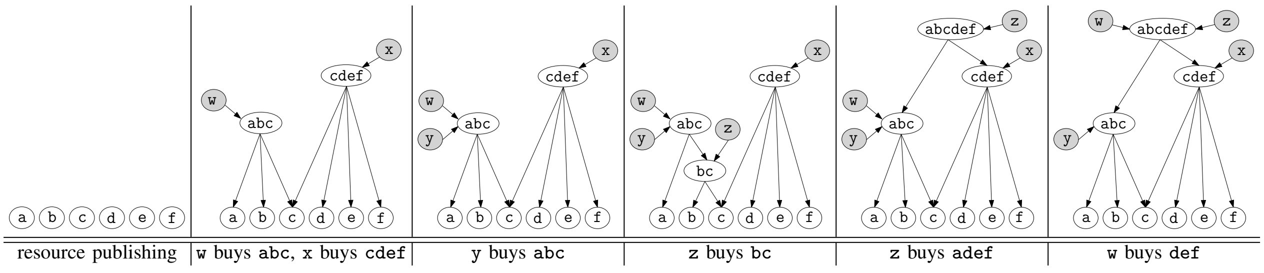


Figure 3. Evolution of a key derivation structure to enforce a sequence of purchases

```
ENFORCE_PURCHASE($p$,$\mathsf{cap}(p)$,$R$,$G(V,E)$)
if $v_p \notin V$ then
    generate $v_p$; $V := V \cup \{v_p\}$
if $v_{\mathsf{cap}(p)} \in V$ then
    $E := E \setminus \{(v_p, v_{\mathsf{cap}(p)})\}$
    if $\nexists p' \neq p$ s.t. $\mathsf{cap}(p) = \mathsf{cap}(p')$ then
        let par be the number of incoming edges of $v_{\mathsf{cap}(p)}$
        let desc be the number of outgoing edges of $v_{\mathsf{cap}(p)}$
        if $(par \times desc) < (par + desc)$ then
            for each $v_{par} \in V : (v_{par}, v_{old}) \in E$ do
                $E := E \setminus \{(v_{par}, v_{old})\}$
                for each $v_{desc} \in V : (v_{old}, v_{desc}) \in E$ do
                    $E := E \setminus \{(v_{old}, v_{desc})\} \cup \{(v_{par}, v_{desc})\}$
            $V := V \setminus \{v_{\mathsf{cap}(p)}\}$
$\mathsf{cap}(p) := \mathsf{cap}(p) \cup R$
if $v_{\mathsf{cap}(p)} \in V$ then $E := E \cup \{(v_p, v_{\mathsf{cap}(p)})\}$
else
    generate $v_{\mathsf{cap}(p)}$
    $V := V \cup \{v_{\mathsf{cap}(p)}\}$
    $E := E \cup \{(v_p, v_{\mathsf{cap}(p)})\}$
    let Desc $\subseteq V$ be the set of vertices over a set of resources $\subset \mathsf{cap}(p)$
    let Cover be a subset of Desc whose resources form a set cover for $\mathsf{cap}(p)$
    for each $v_{cover} \in$ Cover do $E := E \cup \{(v_{\mathsf{cap}(p)}, v_{cover})\}$
    let Par $\subseteq V$ be the set of vertices over a set of resources $\supset \mathsf{cap}(p)$
    let DescCover be the set of vertices reachable from vertices in Cover
    for each $v_{par} \in$ Par do
        ToRemove := $\emptyset$
        for each $v \in$ DescCover $\cup$ Cover do
            if $(v_{par}, v) \in E$ then ToRemove:= ToRemove $\cup \{(v_{par}, v)\}$
        if $|$ToRemove$| \geq 2$ then
            $E := E \cup \{(v_{par}, v_{\mathsf{cap}(p)})\}$
            for each $(v_{par}, v_z) \in$ ToRemove do $E := E \setminus \{(v_{par}, v_z)\}$
```

Figure 4. Procedure managing the purchase of a set of resources

is authorized. Vertex bc then becomes redundant and is removed. The last request authorizes w for all the resources. Since cap(w) belongs to the structure, no vertex is inserted.

V. Counteracting Misbehaviors

Since authorizations are granted upon payment, data owners and processors should trust each other, like in any situation where a vendor sells a product or a service (access to resources, in our case) to a buyer. If access to a resource is granted *before* payment, the owner needs to trust the processor to finalize the payment. If access is granted *after* payment, the processor needs to trust the owner to grant access to the purchased resource(s). Requiring such level of trust is unrealistic in our scenario, and processors and owners might misbehave and get advantages. We then need a solution enabling them to conclude a contract without fully trusting each other. In this section, we first illustrate possible misbehaviors that a malicious party could adopt to gain advantages over the counterpart. We then present our solution to mitigate these risks.

A. Malicious Behaviors

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

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

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

B. Counteracting Approach

We propose an *interaction protocol*, to regulate the interplay between processors and data owners, and an *audit process*, to identify misbehaving parties. The interaction protocol prevents NO_PAYMENT and NO_PAYMENT* misbehaviors, ensuring that no access can be granted without payment. The audit process detects NO_ACCESS and NO_ACCESS* misbehaviors, exposing malicious owners (processors, resp.) that do no grant a purchased access (maliciously claim a purchased access has not been granted, resp.). The combined adoption of these two approaches incentivizes all parties to behave correctly.

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

The interaction between a processor p, willing to purchase a set R of resources, and the owner o of the resources operates according to the protocol in Figure 5:

1) p contacts o (off-chain) communicating the set R of resources she wants to buy;
2) if p and o have not yet interacted, o generates a key k_p for p and sends it to p;
3) upon receiving k_p, p signs it with her secret key $priv_p$, and sends the signed key (denoted $[k_p]_{priv_p}$) to o;
4) o prepares a smart contract, dictating that "upon receiving `price` from p for R, $\mathsf{cap}(p) := \mathsf{cap}(p) \cup R$, and the token catalog $\mathcal{T}$ is updated in such a way to allow p to derive the keys for the resources in $\mathsf{cap}(p)$";
5) o deploys the contract on the blockchain;
6) p accesses the contract and signs it, automatically triggering the payment for `price`;
7) o updates the key derivation structure as required by the executed contract (see Section IV);
8) o updates $\mathcal{T}$ on the blockchain.

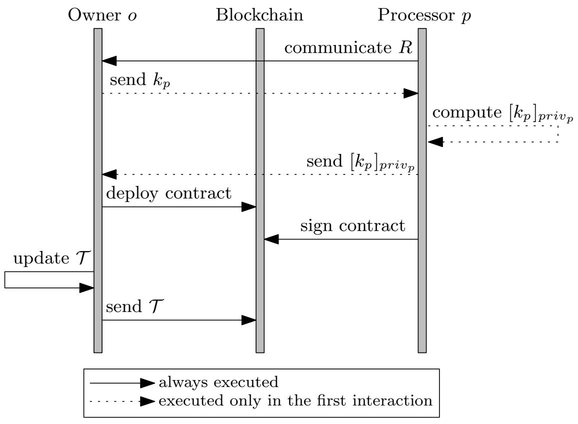


Figure 5. Interaction protocol

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

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

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

```
AUDIT($p$,$o$)
retrieve $\mathsf{cap}(p)$
retrieve $t_{p,\mathsf{cap}(p)}$ and $l_{\mathsf{cap}(p)}$ from $\mathcal{T}$
if $t_{p,\mathsf{cap}(p)}$ = NULL or $l_{\mathsf{cap}(p)}$=NULL then
  return 'NO_ACCESS misbehavior'
retrieve $[k_p]_{priv_p}$ from $o$
if signature verification of $[k_p]_{priv_p}$ fails then
  return 'NO_ACCESS misbehavior / $o$ not collaborating'
compute $k_{\mathsf{cap}(p)} = h([k_p]_{priv_p}, l_{\mathsf{cap}(p)}) \oplus t_{p,\mathsf{cap}(p)}$
derive all the resource keys $k_r$ derivable from $k_{\mathsf{cap}(p)}$
for each $r \in \mathsf{cap}(p)$ do
  $r$ := decrypt$(E(r), k_r)$
  if decryption fails then
    return 'NO_ACCESS misbehavior'
return 'NO_ACCESS* misbehavior'
```

Figure 6. Pseudocode of the audit process

Given the identity of the processor p and of the owner o involved in the audit process, the auditor checks whether the current token catalog $\mathcal{T}$ enables p to derive the keys for the resources in $\mathsf{cap}(p)$ to discriminate between NO_ACCESS and NO_ACCESS*. Figure 6 illustrates the audit process. The auditor first needs to query the public ledger maintained on-chain to obtain: the capability list $\mathsf{cap}(p)$ of the processor, the token for deriving key $k_{\mathsf{cap}(p)}$, and label $l_{\mathsf{cap}(p)}$. If the token (or the label) does not exist, the auditor signals a NO_ACCESS misbehavior. In fact, the derivation structure cannot allow p to derive the keys for the resources in $\mathsf{cap}(p)$ (i.e., the owner ignored the payment). Otherwise, the auditor retrieves $[k_p]_{priv_p}$ from the owner and, using the catalog, derives all the encryption keys reachable from $[k_p]_{priv_p}$. To verify that $[k_p]_{priv_p}$ is the key that p and o exchanged in the interaction protocol (Figure 5), the auditor checks its signature. If signature verification fails, either o has defined/updated the derivation structure starting from the wrong key (and hence p cannot access the resources in her capability list), or it is not participating honestly in the audit process (i.e., she returned a different key from the one agreed with p). In both cases, the auditor signals a NO_ACCESS misbehavior, exposing a misbehavior of the owner. On the contrary, if signature verification succeeds, the auditor derives $k_{\mathsf{cap}(p)}$ and the keys of the resources in $\mathsf{cap}(p)$. The auditor can then try to decrypt all resources in $\mathsf{cap}(p)$ using these keys. If decryption fails (because for at least one resource the related key is not derivable or incorrect), the auditor again signals a NO_ACCESS misbehavior. Otherwise, if decryption succeeds, the auditor returns a NO_ACCESS* misbehavior, since the owner respected her obligation and hence the processor is dishonestly accusing the owner of misbehavior. Note that the audit process could expose the plaintext content of resources to the auditor, when the processor maliciously accuses the data owner of misbehavior. However, a malicious processor can disclose the purchased resources to any subject (hence including the auditor) independently from the audit process, so the process itself does not introduce additional disclosure risks. Also, thanks to our audit process, the misbehavior of the processor is revealed. Therefore, we expect processors not to dishonestly blame a misbehavior on an honest owner, as this would decrease their reputation.

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

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

VI. Discussion

The combined adoption of selective encryption and of an approach based on blockchain, smart contracts, and an audit process for regulating the interactions between data owners and processors can set a first step (out of many) towards the enforcement of the transparent processing requirement of the EU GDPR. Transparent processing of data implies, among other aspects, to log all events related to data processing and sharing, and to enable the control that the processing itself is performed according to the policy set by the data subject (the data owner, in our scenario) [6]. We fulfill these requirements by ensuring that: *i)* each resource is shared only with processors authorized by the owner; and *ii)* sharing is securely logged on-chain, producing a verifiable trail of sharing history. This ensures that the data owner knows and can prove, at any time, who is able to access her resources, and that each processor is able to prove that all accessed resources were authorized. Also, since we store both the authorization policy and the token catalog on-chain, their updates leave a permanent trace. Hence, not only is it possible to verify the enforcement of the most up-to-date policy, but also past versions of the token catalog can be checked for verifying the correct enforcement of a former one. Also, since resources are not stored on-chain, the owner can always delete them in accordance to the regulation.

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

VII. Related Work

The adoption of distributed ledgers (such as blockchain) and of smart contracts has recently been investigated for exchanging data and/or enforcing access control (e.g., [7], [8], [9], [10], [11], [12]). In [7], the authors propose a solution for auditable sharing of private data on blockchains, which however assumes a (collective) authority for enforcing access restrictions. The proposal in [8] shares with ours the adoption of encryption for protecting data, but does not consider purchases and subsequent possible misbehaviors. The problem of data trading has been investigated in [9], which however adopts Bitcoin transactions and does not support fine-grained authorizations. The approach in [12] focuses on ensuring fairness of resource exchange, without explicit reference to data markets and their need for fine-grained authorizations. A blockchain-based access control model is proposed in [10], but the possibility of a processor to transfer her access rights to another processor does not fit our scenario. The proposal in [11] protects data using encryption and on-chain storage of pointers to data, while we decouple data storage and blockchain. Other relevant examples of blockchain applications include e-auctions [13] and data provenance architectures [14].

Another line of work close to ours is related to the enforcement of access control in data outsourcing. Related approaches have explored the adoption of selective encryption (e.g., [2], [3], [15]), on which we build part of our solution. These proposals however do not consider the purchase requirements of our reference scenario, and typically focus on enforcing authorization policies defined and updated according to the wishes of the data owner.

VIII. Conclusions

We presented an approach combining selective encryption, blockchain, and smart contracts to enable data owners to leverage data markets to monetize their data in a controlled way. Our approach is complemented by an audit process counteracting misbehaviors in case of dishonest subjects. An interesting direction for extensions concerns the secure development of smart contracts.

Acknowledgment

This work was supported in part by the EC within the H2020 Program under grant agreement 825333 (MOSAICrOWN).

References

[1] C. Ardagna, V. Bellandi, M. Bezzi, P. Ceravolo, E. Damiani, and C. Hebert, "Model-based big data analytics-as-a-service: Take big data to the next level," *IEEE TSC*, 2018, pre-print.

[2] M. Atallah, M. Blanton, N. Fazio, and K. Frikken, "Dynamic and efficient key management for access hierarchies," *ACM TISSEC*, vol. 12, no. 3, pp. 18:1–18:43, January 2009.

[3] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Encryption policies for regulating access to outsourced data," *ACM TODS*, vol. 35, no. 2, pp. 12:1–12:46, April 2010.

[4] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution," *arXiv preprint arXiv:1804.05141*, 2018.

[5] S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati, "Practical techniques building on encryption for protecting and managing data in the cloud," in *The New Codebreakers*, Springer, 2016.

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

[7] E. Kokoris-Kogias, E. C. Alp, S. D. Siby, N. Gailly, L. Gasser, P. Jovanovic, E. Syta, and B. Ford, "CALYPSO: Auditable sharing of private data over blockchains," Cryptology ePrint Archive, 2018/209, Tech. Rep.

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

[9] S. Delgado-Segura, C. Pérez-Sola, G. Navarro-Arribas, and J. Herrera-Joancomartí, "A fair protocol for data trading based on bitcoin transactions," *FGCS*, 2017, pre-print.

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

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

[12] S. Dziembowski, L. Eckey, and S. Faust, "FairSwap: How to fairly exchange digital goods," in *Proc. of CCS*, Toronto, Canada, 2018.

[13] S. Wu, Y. Chen, Q. Wang, M. Li, C. Wang, and X. Luo, "CReam: A smart contract enabled collusion-resistant e-auction," *IEEE TIFS*, vol. 14, no. 7, pp. 1687–1701, July 2019.

[14] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, "ProvChain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability," in *Proc. of CCCGrid*, Madrid, Spain, 2017.

[15] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati, "Mix&Slice: Efficient access revocation in the cloud," in *Proc. of CCS*, Vienna, Austria, 2016.