DEFS - Data Exchange with Free Sample Protocol

RAFAEL GENÉS-DURÁN, *UPC*

JUAN HERNÁNDEZ-SERRANO, *UPC*
OSCAR ESPARZA, *UPC*

MARTA BELLÉS-MUÑOZ, *UPF*

JOSÉ L. MUÑOZ-TAPIA, *UPC*

15 June 2021

Abstract

Distrust between data providers and data consumers is one of the main obstacles hampering digital-data commerce to take off. Data providers want to get paid for what they offer, while data consumers want to know exactly what they are paying for before actually paying for it. In this article, we present a protocol that overcomes this obstacle by building trust based on two main ideas. First, a probabilistic verification protocol, where some random samples of the real dataset are shown to buyers in order to allow them to make an assessment before committing any payment; and second, a guaranteed, protected payment process enforced with smart contracts on a public blockchain, that guarantees the payment of the data if and only if the data provided meets the agreed terms, and that refunds honest players otherwise.

Keywords: Blockchain, Marketplace, Privacy, Fairness, Conflict resolution, Non-repudiation, Probabilistic verification.

1 Introduction

The use of data has increasingly become a crucial factor in the success of businesses. Research has shown that proper use of big data techniques helps to identify new insights, optimize operating processes and make better and faster decisions [1]. In this context, ecosystems have grown to fulfill the data needs of diverse actors, such as data suppliers, data custodians or data aggregators. As a result, businesses not only collect and analyse the data they generate, but increasingly rely on third party data to enhance its business value. The necessity of exchanging data between different parties gives rise to an ecosystem that has an inherent regulatory complexity and a need for privacy. In general, making proper data agreements is not easy, specially the task of valuing data and convincing customers of their value without giving them away [2]. The creation of marketplaces addresses many of these problems. Allowing providers and consumers to deal with common interests in a platform where both parties can meet each other and trade information solves the integration problem of connecting consumers and providers.

In this article, we focus on the problem of convincing consumers of data value, which can be seen as a form of lack of trust towards data providers. Traditionally, this problem could not be solved without previously establishing certain confidence between parties. This fact represents an entry barrier to new providers in the market, hurting competence and thus, reducing utility for consumers. Achieving the exchange of virtual products between many parties while minimizing risks is the main goal of virtual commerce. In order to exchange value safely, it is essential to ensure that consumers get the product they pay for and that providers get paid. These two things are often carried out without any strict protocols and guaranteed just by existing trust. Typically, counterparties that know each other from previous experience or that are aligned with future interests, are confident that no intent to scam will be made by the other party, since confidence is often more beneficial than gains from fraud.

Nonetheless, when stronger assurance than that is needed, it is a common practice to use a trusted third-party (TTP) to whom all parties trust to guarantee that the process is carried out correctly by all individuals involved. TTPs solve the crucial aspect of minimizing risks, but ensuring its viability entails an extra cost for all parties. Moreover, centralizing interactions between businesses via the TTP generates a single point of failure that could produce critical delays and denial of services. Distributed Ledger Technologies (DLTs) can be seen as a paradigm shift when it comes to the need of TTPs. Using DLTs, all participants in the network can maintain a set of synchronized data (who owns what) without the need for a central authority (TTP) guaranteeing integrity, fairness and data availability. In addition, recent studies have shown that replacing TTPs by DLTs represents an important optimization of time and overall costs [3].

In this article we present DEFS, a protocol that addresses the lack-of-trust problem between providers and consumers in a data trade. Our protocol preserves the security, privacy and fairness standards that marketplaces should guarantee, and it also includes the capability of checking some sample portions of the dataset before committing to purchase to enhance the trust of the consumers in the data value.

The article is organized the following way: In Section 2, we give an overview of the technologies used in our protocol. Section 3 contains the state of the art about decentralized data marketplaces. In Section 4, we explain our protocol. First, we give a general overview and afterwards, we provide a detailed description of each step of the protocol. In the following section, Section 5, we present a security analysis and finally, we conclude in Section 6.

2 Background

2.1 Distributed Ledger Technologies

The main technology to build a public ledger is a blockchain network. In a blockchain network users can run a blockchain node to send their transactions or use some available node that allows them to do so. Then, in a distributed way, the blockchain network can create a unique sequence of ordered transactions. In more detail, the network creates a chain of blocks following a consensus algorithm to order transactions [4]. A block contains several transactions, and an important property of these systems is that, once the consensus algorithm definitively accepts a block, all nodes get to know this block and it becomes impossible to manipulate or delete it [5].

In a blockchain network, users can own one or more accounts. Accounts are identified via a public identifier (usually derived from a random public key using a hash function). New blockchain accounts can be created by simply generating a pair of asymmetric keys and deriving the account identifier from the public key. In general, account identifiers are not directly linked with any user data, so they can be considered pseudo-anonymous identifiers. Transactions carry the source account identifier and a destination account identifier, and they are all digitally signed using the private key of the source account. All the nodes that form the blockchain network see the same state (also known as world state) that results from executing all the transactions in order.

2.2 Smart Contracts

Some blockchains not only allow executing regular transactions that modify the cryptocurrency balances on the ledger but also have the capability of deploying and executing public and auditable programs called smart contracts. Smart contracts have their own state and in their code we can define the business logic we want to process transactions. Once a smart contract is deployed in the blockchain network, its code is replicated on every node, and consequently, these programs have the same availability and integrity as regular transactions. The Ethereum [6] mainnet is a good candidate to implement our proposal because it is a public blockchain, capable of running smart contracts, and it is the platform of choice for many developers for implementing Decentralized Applications (DApps).

2.3 Merkle Hash Trees

A Merkle Hash Tree (MHT) is an authenticated data structure where every leaf node of the tree contains the cryptographic hash of a data block and every non leaf node contains the concatenated hashes of its child nodes [7]. MHTs allow to link a set of data to a unique hash value, the Merkle hash tree Root (MR), allowing efficient and secure verification of consistency and content of large sets of data.

Figure 1 contains an example of a MHT with 8 leaves. To show that a certain value is stored in a leaf of the MHT, one can create a Merkle Proof (MP), which consists of a list of the additional nodes required to compute the root of the tree. For instance, a MP showing that h_3 is stored in the MHT from Figure 2 would consist of the nodes

$$MP(h_3) = \{h_2, h_{01}, h_{4567}, h_{01234567}\}.$$

Note that with h_3 and the first three nodes of this list anyone can compute the root of the tree. If the root matches $h_{01234567}$, then the proof is valid proof of membership for h_3 in the tree.

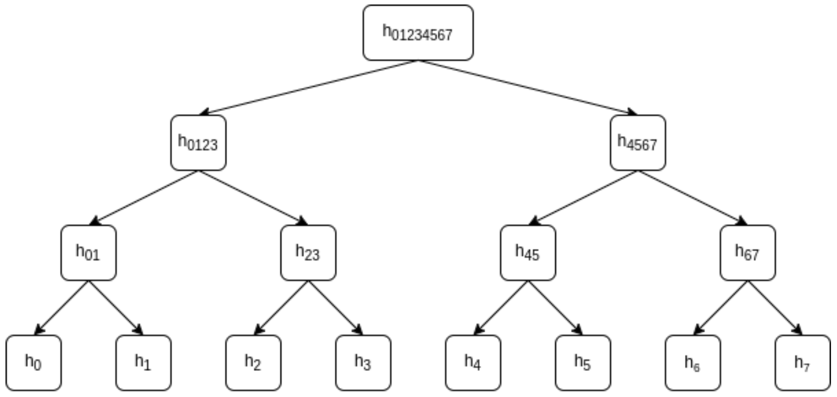

Figure 1: MHT of 8 leafs.

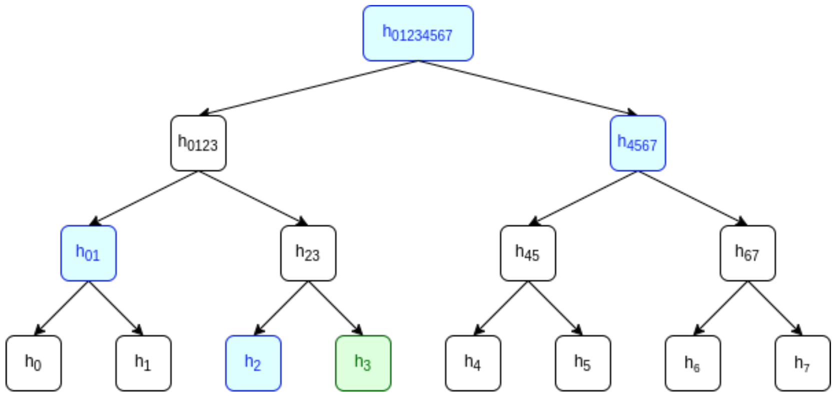

Figure 2: The MP of h_3 is the set $\{h_{01234567}, h_{4567}, h_{01}, h_2\}$, which contains the nodes needed to compute the MR of the tree.

The security of a MP reduces to the collision resistance of the underlying hash function [8]. For this reason, we assume the hash function H used to build

MHTs is cryptographically secure. That is, that the probability of finding a preimage or a hash collision is negligible [9].

3 State of the Art

Traditional data marketplaces build trust by requiring identifying information to the different stakeholders. This information and the process to get it is typically known as Know Your Customer (KYC). The belief is that KYC constitutes an entry barrier to fraudsters, since the KYC data would help victims to recur to law enforcement if tangible fraud is committed. However, marketplaces with KYC lead to regulatory complexity and are, in general, difficult to operate with. This fact hinders the growth of digital-data trading and can make stakeholders feel that their privacy is violated. In addition, law execution is slow and in some cases might be even useless. For the previous reasons, there have been attempts in the research community to design new approaches to mitigate these limitations with technology and not regulations. In this context, decentralized marketplaces have arisen as a solution to enhance security, sovereignty and trust in data exchanges [10–12].

One interesting initiative is GAIA-X [13], which is an European project created to develop the foundations for a federated open-data infrastructure connecting both classical architectures with decentralized infrastructures in order to build a transparent ecosystem for the end users taking advantage of the decentralized benefits.

One of the main technologies that is fostering data marketplaces is the Internet of Things (IoT), which generates huge amounts of data from sensors and devices. The increasing necessity of monetizing these data is also pushing research. In literature, we can find several works that propose decentralized marketplaces for IoT using distributed ledger technologies to enhance the data exchanges with transparency, trust and integrity [14–16]. Among others, decentralized marketplaces are being implemented in new disruptive scenarios such as artificial intelligence [17], smart cities [18, 19], and connected cars [20]. In fact, the value of data is becoming more important to the business interactions, which is reflected in the new technologies and their necessity to generate this new era of decentralized marketplaces.

A remarkable example of a decentralized data trading solution is presented in [21]. As in our protocol, the data on sale are not stored on the blockchain but in some external (and possibly distributed) storage platform. Similar to our protocol, the proposed solution symmetrically encrypts data on sale and uses a MHT of cryptograms to register the associated trades on the blockchain. However, the solution proposed not only requires to generate symmetric cryptograms but also the need of asymmetrically sign each of these cryptograms. Additionally, authors propose to use Plaintext Checkable Encryption (PCE) [22] to check on-chain that cryptograms have been correctly encrypted. In our protocol, we avoid using asymmetric encryption, which is much slower than symmetric encryption. In DEFS, we achieve a faster and easier solution by providing structure to symmetric keys and generating a MHT with these keys to allow solving disputes with regards to the data encryption. In addition, [21] considers three roles: data buyers, data sellers, and miners. The main problem of involving directly miners in the implementation of the solution is that then, the mining software needs to be modified, which is, in general, not a trivial thing to do. Mining software is extremely subtle, since any error in an implementation can lead to a lack of consensus in the network. In our protocol, we also consider the roles of data buyers and data sellers but the role of miners is abstracted, and we use the API provided by smart contracts which is a much easier and safer way of implementing the logic of data trades in the blockchain.

Another remarkable implementation of a decentralized data trading solution is presented in [23], where the authors present SDTE, a secure blockchain-based data trading ecosystem. As our protocol, SDTE tries to mitigate the existence of dishonest parties in data exchanges. However, SDTE focuses on an scenario in which the buyer does not need to have access to a complete dataset but it only needs the findings from the data analysis. For this case, SDTE proposes a data processing-as-a-service, where the buyer is paying for the analysis of the seller's dataset. SDTE is build using an Intel's SGX-based secure execution environment to protect the data processing, the source data and the analysis results. As we will show in the following section, DEFS is not designed as a data processing-as-a-service but as a data exchange-as-a-service. In the latter, the seller wants to buy the complete dataset not computed data. For this scenario, DEFS provides a probabilistic verification protocol and a conflict resolution protocol that is guaranteed and supported by a smart contract.

4 Data Exchange Protocol

In this section we introduce DEFS, a protocol that addresses the problem of data trading between a provider and potential consumers using a smart contract deployed in the blockchain as a broker. To mitigate gender issues when referring to a single provider and a single consumer, we will assume the provider is a woman and the consumer a man.

As we explained before, the use of DLTs can replace the role of TTPs in payment processes. When using DLTs, participants in the network can maintain synchronized data and share payment information without the need of a central authority, guaranteeing this way the integrity, fairness and availability of the data. In this manner, DEFS makes use of a smart contract to preserve the security and privacy standards that marketplaces should guarantee.

Another gap to cover in this data trading scenario is generating trust between data consumers and data providers. Here it comes the novelty of DEFS: our proposed data exchange protocol is designed with the capability of checking random samples from the dataset, so that consumers are able to infer if the complete dataset is worth to be paid for, enhancing the trust from the consumer's side. On the other side, the smart contract acts as a broker during the payment procedure, ensuring providers that they will receive the payment for the data they exchanged.

4.1 Protocol Overview

First, we give a general overview of DEFS, and in Section 4.3, we describe in greater detail all the steps the entities involved (consumer, provider and smart contract) should follow. We assume that, before starting the protocol, a data provider advertises her data to the public using off-blockchain means, such as a data marketplace. Then, a consumer interested in a particular dataset contacts the provider, who starts the DEFS protocol to perform the data exchange and payment. To prevent potential extensive leaks of the data, it is important that the DEFS protocol is executed independently per each individual consumer. DEFS consists of three different phases:

1. **Protocol preparation**: in this initial phase, the provider prepares not only the data to be exchanged, but also all the parameters and cryptographic material necessary to demonstrate that the data exchange is secure and private. More specifically, the provider:

    - Divides the complete dataset in portions. These portions are chosen randomly (not consecutively) from the dataset.
    - Generates a seed to generate symmetric cryptographic keys.
    - Uses these keys to create a MHT, whose root can be used to check the correctness of this cryptographic material.
    - Encrypts a random permutation of the data portions with the keys, obtaining an encrypted and randomized version of the whole dataset.
    - Creates another MHT using the hashes of these cryptograms as leaves, whose root can be used to verify the correctness of the cryptograms generated.
    - Deploys a smart contract in the blockchain that includes among other information, the roots of the previous trees.

    If a consumer has interest in obtaining the dataset, the protocol continues as follows:

    - The consumer receives the whole dataset encrypted but it cannot be decrypted at that very moment.
    - The consumer queries the smart contract to obtain the root of the tree of cryptograms and verifies that all the cryptograms belong to this tree.

    As previously stated, this is only a brief summary of the steps to follow in this phase. A more exhaustive explanation of the protocol preparation phase can be found in Section 4.3.3. At this point, all entities (consumer, provider and smart contract) are ready to start the protocol execution phase, in which the consumer will have access to the complete dataset and will perform the payment.

2. **Protocol execution**: in this phase, the consumer gets some samples of the dataset (for free) to evaluate if it is worth to pay for the whole set, and if so, he will obtain the dataset and the provider will get paid:

   - The consumer chooses at random some sample portions to be revealed. Note that, the provider committed the shuffled encrypted data at the very beginning of the protocol. Since the consumer requests random samples, neither consumers nor providers have control over the samples that will be revealed.
   - The provider discloses the keys for those samples, so the consumer can evaluate the quality of the dataset.
   - If the consumer is not convinced, the protocol ends here. However, he decides that it is worth paying for the dataset, he commits the payment to the smart contract.
   - The provider is asked to publish the seed (that will disclose all the encryption keys) in the smart contract.
   - If the consumer is able to properly decrypt the dataset, after a timeout, the provider gets paid and the protocol ends.
   - If the consumer is able to prove that there were problems with the previous procedure, he starts a conflict resolution phase to obtain a refund.

   A more exhaustive explanation of the protocol execution phase can be found in Section 4.3.4.

   The following phase will only be needed in case the consumer considers that he has been cheated on.

3. **Conflict resolution***: this phase is optional and it only takes place if the consumer detects a provider misbehaviour. The conflict resolution can end with a refund if the consumer is able to demonstrate one of the following misbehaviours:

   - A key is not properly generated.
   - A cryptogram does not have the proper format when decrypted.

   A more exhaustive explanation can be found in Section 4.3.5.

4.2 Protocol Properties

The main properties provided by our protocol are the following ones:

1. **Data samples evaluation**. The consumer gets a free set of fair samples of the data being traded before paying. The protocol ensures that neither the consumer nor the provider are able to manipulate the chosen data or select specific samples.

2. **Payment guarantees**. The provider gets paid if and only if the consumer has access to the whole set of data. That is, the consumer can not get the data without paying for it and the provider does not get paid without disclosing the data.

3. **The solution is cost-efficient**. Due to high fees on public ledgers, DEFS minimizes the amount of data stored on the ledger, which is also independent of the quantity of data traded. This way, both the amount of data stored and the number of interactions with the distributed ledger is constant.

4. **Non-repudiation**. The DEFS protocol ensures that any party involved in the exchange is not able to cancel and/or deny the data exchange once an agreement is made.

5. **Liveness**. The different timeouts guarantee that the protocol reaches a final state, even when one of the parties quits in advance.

4.3 The DEFS Protocol

In this section we describe the DEFS protocol. We establish the notation in Section 4.3.2. The procedure before initiating the exchange is detailed in Section 4.3.3. Then, the interactions to do a fair transaction are explained in Section 4.3.4. The conflict resolution is detailed in Section 4.3.5. Finally, we present the state diagram of the smart contract in Section 4.3.6.

4.3.1 Requirements

DEFS protocol assumes that measures to meet the following requirements are already in place:

Secure off-chain channel between provider and consumer: It is assumed that the off-chain channel between consumer and provider is end-to-end protected. This requirement can be easily met by using the widely supported TLS protocol – e.g. with HTTPS. TLS only requires the server to hold a valid certificate (and its complementary private key) in order to create the secure channel. That is to say, consumers just need a valid TLS client, which is implemented by default in most programming languages, application frameworks, and/or web browsers.

Validation of data blocks' format: It is assumed that consumers can verify received data blocks according to a previously agreed schema. The process that verifies that a data portion meets a predefined format is usually called a validator. Validators are used by many technologies to check received responses before processing them. In object-oriented programming, this process is usually done by trying to parse the response as a given type of object, which will throw an error if it does not. There are also specific standards with well-known implementations, such as JSON-LD [24], that help define and validate specific data schemas.

Identification system: The DEFS protocol releases random samples of the dataset to potential consumers before they commit to pay for the entire dataset. However, there is a risk of an attacker using multiple identities to retrieve a representative portion of the dataset for free. In this context, DEFS assumes that there are off-chain solutions run by the providers which could effectively limit the amount of identities an attacker could take. A known example is binding the identity to an e-mail account or a mobile phone. The provider should decide the most suitable authentication method depending on the price of the traded data and the type of consumers. For example, in some cases, authenticating with an e-mail can be enough. In other scenarios, e-mail might not be enough because it is not hard to generate multiple "identities" based on different e-mail accounts. In the latter case, providers might require authenticating with a mobile phone or even with both factors. In some specific cases, authentication could involve more factors, such as physical key generators, smart cards, etc.

4.3.2 Notation

The notation of the DEFS protocol is summarized in Table 1.

4.3.3 Protocol Preparation

In Figure 3 we detail the interactions between provider, consumer and smart contract during the preparation phase of the DEFS protocol.

The steps of this phase are enumerated and explained just below:

1. **Consumer→Provider**: Request *data*.

   The protocol starts with the consumer's interest of a dataset. Through the marketplace, the consumer requests some offered *data* to the provider. Note that, each time the consumer desires a dataset, it requires a new instance of the DEFS protocol.

2. **Provider**: Set $id, p, n, v, collateral$.

   In this step the provider has to decide the main parameters associated to the dataset. These parameters are: the identifier of the data exchange (id); the price of the dataset (p); the number of portions in which the dataset

Notation	**Description**
collateral	Price that the provider must pay in order to ensure fairness.
$C = \{c_0...c_{n-1}\}$	Encrypted data portions (cryptograms) such that $c_i = E_{K_i}(d_i)$.
$D = \{d_0...d_{n-1}\}$	Data portions.
e	Index of an invalid cryptogram.
id	Data-exchange identifier.
$K = \{k_0...k_{n-1}\}$	Encryption keys for data portions, such that $k_i = hash(s + i)$.
$MHT(C)$	Merkle hash tree of cryptograms.
$MHT(K)$	Merkle hash tree of keys.
MRC	Root of the Merkle hash tree of cryptograms.
MRK	Root of the Merkle hash tree of keys.
MPC_i	Merkle proof of a cryptogram with index i.
MPK_i	Merkle proof of an encryption key with index i.
n	Number of data portions.
p	Price of the dataset.
$R = \{r_0...r_{v-1}\}$	Set of indexes of the sample portions to be revealed.
s	Seed. Random number for key generation.
v	Number of sample portions to be revealed.

Table 1: Notation for the DEFS protocol.

will be divided (n); the number of sample portions to be revealed before the payment (v); and finally, the associated amount of cryptocurrency to ensure a complete refund in case a conflict resolution ends in favour of the consumer ($collateral$).

How to choose v depends on the identification system in use (see Section 4.3.1) and the resilience from providing for free a representative part of the dataset to one or multiple attackers. The analysis in Section 5.3 shows how to properly choose v based on the size of the dataset and the estimated amount of identities an attacker can hold.

3. **Provider**: Generate s.

   The provider should generate the symmetric encryption keys in a way that, in case of disclosing some of them, the consumer will not be able to derive any other key (or the whole set). In addition, the consumer must be able to easily derive all the keys when it agrees to buy the dataset. A simple way to achieve these features is by generating an initially private seed and use a cryptographic hash function to compute the whole set of keys. For that reason, the seed s is calculated using a random number generator.

4. **Provider**: Compute K=[$k_0...k_{n-1}$] — $k_i = h(s+i)_i$∀i ∈{0...n-1}.

   The provider has to compute a set of n symmetric encryption keys (K=[$k_0..k_{n-1}$]). In DEFS, we compute each key as the hash function, for instance Kec-

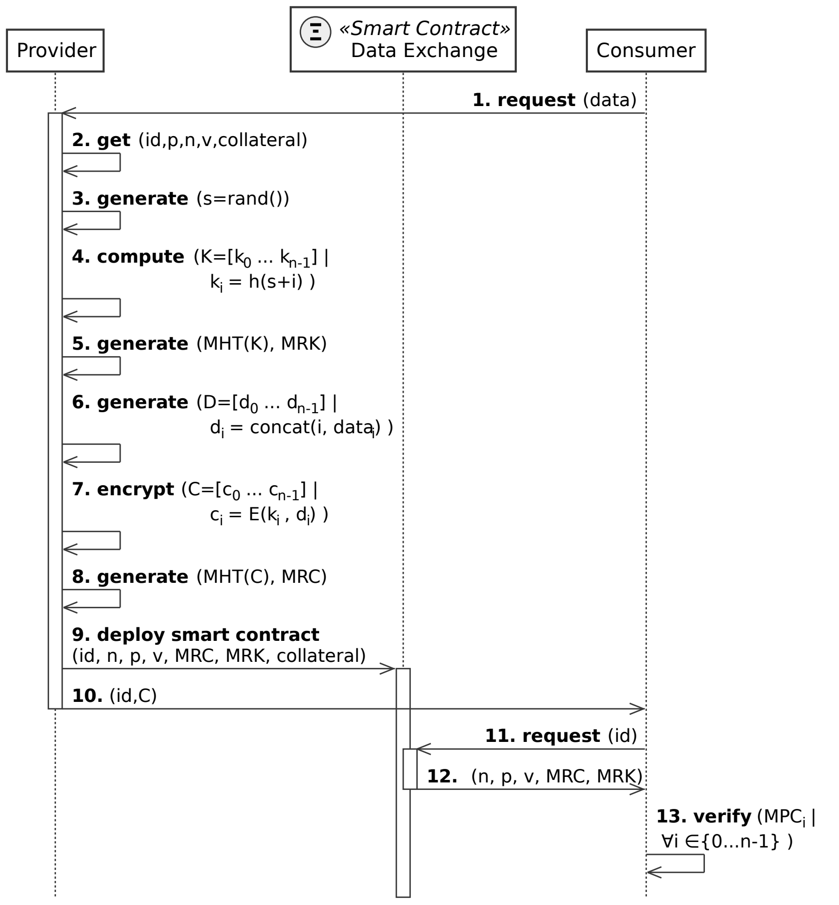
Provider
«Smart Contract» Data Exchange
Consumer
1. request (data)
2. get (id,p,n,v,collateral)
3. generate (s=rand())
4. compute (K=[k0 ... kn-1] | ki = h(s+i))
5. generate (MHT(K), MRK)
6. generate (D=[d0 ... dn-1] | di = concat(i, datai))
7. encrypt (C=[c0 ... cn-1] | ci = E(ki , di))
8. generate (MHT(C), MRC)
9. deploy smart contract (id, n, p, v, MRC, MRK, collateral)
10. (id,C)
11. request (id)
12. (n, p, v, MRC, MRK)
13. verify (MPCi | ∀i ∈{0...n-1})

Figure 3: Protocol preparation: sequence diagram.

cak256, of the sum of the seed s and the index i using the following formula:

$$k_i = hash(s + i))\forall i \in \{0...n - 1\}.$$

This construction has the expected properties: without the seed s the consumer cannot derive any other key, but once the seed is known, it is easy for the consumer to calculate the whole sequence of keys.

5. **Provider**: Generate $MHT(K)$.

   The provider builds a binary MHT for the set of keys, which is going to be used to generate the proof of the correctness of the keys used to encrypt the data portions. We denote the MHT of encryption keys as $MHT(K)$, its root MRK and we refer to a membership proof of a leaf i as MPK_i.

Figure 4 shows an example of a $MHT(K)$. An exhaustive explanation about the algorithm to construct these trees can be found in [25]. Here we are just going to include a brief summary of this algorithm. To construct the $MHT(K)$, keys must be sorted by using their indexes, from 0 to $n-1$. The leaves of the tree are calculated by hashing the keys in their respective position (hashed keys). The rest of the intermediate nodes in upper levels are just calculated by hashing the concatenation of the lower left and right nodes of the same branch. The tree construction continues until reaching the top level, in which we obtain the MRK. Notice that the MRK is the digest of the complete key set K, and it can be used as a proof of its correctness. It is also important to remark that the algorithm to construct this tree should be public, and all the entities have to use the same algorithm, because any change in the keys or in the order of constructing it will cause an avalanche effect that will make the root MRK to be completely different.

6. **Provider**: Generate D=[$d_0 ... d_{n-1}$] — $d_i = concat(i, data_i)$∀i ∈{0...n-1}.

   Now, using the pre-existing data to be exchanged, the provider has to build an array ($D = [d_0 ... d_{n-1}]$) with the portions where each d_i has the corresponding data and the index as a header $d_i = concat(i, data_i)$. This format is going to allow the smart contract to determine if the cryptograms d_i have been properly generated. Also, it is important to note, that each data portion $data_i$ contains a group of random registries, not consecutive ones. To do that, registries are sorted using an external random generator tool provided by the marketplace, which should be open source and auditable to avoid duplicated entries in the same portion. This will avoid a potential attack where the consumer replicates several data requests to obtain free samples without committing any payment. More details about potential attacks can be found in Section 5.3.

7. **Provider**: Encrypt data. C=[$c_0 ... c_{n-1}$] — c_i=$E_{k_i}(d_i)$∀i ∈{0...n-1}.

   Now the provider is able to encrypt all the portions of the dataset d_i, and obtain the set of cryptograms C.

8. **Provider**: Generate $MHT(C)$.

   The provider builds a binary MHT for the set of cryptograms, which is going to be the proof of their correctness. We denote the MHT of cryptograms as $MHT(C)$, its root MRC and we refer to a membership proof of a leaf i as MPC_i. Figure 5 shows an example of an $MHT(C)$. The algorithm of the $MHT(C)$ is the same as the $MHT(K)$, but just changing the information used to construct it. Cryptograms are sorted by using their indexes, from 0 to $n-1$. The leaves of the tree are calculated by hashing the cryptograms in their respective position (hashed encrypted data). The rest of the intermediate nodes in upper levels are just calculated by hashing the concatenation of the lower left and right nodes of the same branch. The tree construction continues until reaching the top level, in which we obtain the Root of the Merkle hash tree of cryptograms (MRC). Notice that the MRC is the digest of the complete set C, and it can be used as a proof of its correctness.

9. **Provider→SC**: Deploy smart contract with parameters (id, n, p, v, MRC, MRK, $collateral$).

   Next, the provider deploys a smart contract in a ledger that stores the data-exchange identifier (id), the total number of portions (n), the number of sample portions to be revealed before the payment, the price (p) of the dataset and the amount of cryptocurrency to assure the fairness from the provider on the conflict resolution process ($collateral$), and the root of both Merkle hash trees (MRC,MRK). These roots will allow proving whether an element is or is not a cryptogram or a key and its position in the MHT. Using this, the system is able to efficiently assure the consumer that the provider cannot alter the committed dataset.

10. **Provider→Consumer**: Send id and the complete set of cryptograms (C).

    Finally, the provider delivers the smart contract address, the data-exchange identifier id, and the complete set of cryptograms C to the consumer. Obviously, it would be totally impractical to exchange that amount of data using the ledger as storage. Instead, the exchange of cryptograms between the provider and consumer is done off-blockchain. Notice also that, since the seed will be public at the end of the process, all the off-blockchain traffic must have been exchanged using a secure channel.

11. **Consumer→SC**: Request data from id.

    At this point, the consumer has the id which is related to the deployed smart contract and can read the values set by the provider in step 9.

12. **SC→Consumer**: Reply with values n, p, v, MRC, MRK.

    Now, the consumer has the total number of portions (n), the number of samples he can obtain before committing the payment (v), the price of the data (p), the Merkle roots of both trees (MRC,MRK), and the complete set of cryptograms (C).

13. **Consumer**: Compute $MHT(C)$ and verify MRC.

    As the consumer has the complete set of cryptograms C, he has the capability and responsibility to re-generate $MHT(C)$ to verify that the root MRC calculated is coherent with the one at the smart contract. If so, the consumer knows that all cryptograms c_i $\forall$i $\in${0...n-1} were properly generated and match the MRC. The consumer is responsible for verifying the $MHT(C)$ at this very moment, and if he continues with the protocol tacitly accepts the correctness of the generation of the cryptograms. This means that, in case of a later conflict resolution, the consumer cannot argue that the cryptograms were wrongly generated to get a refund.

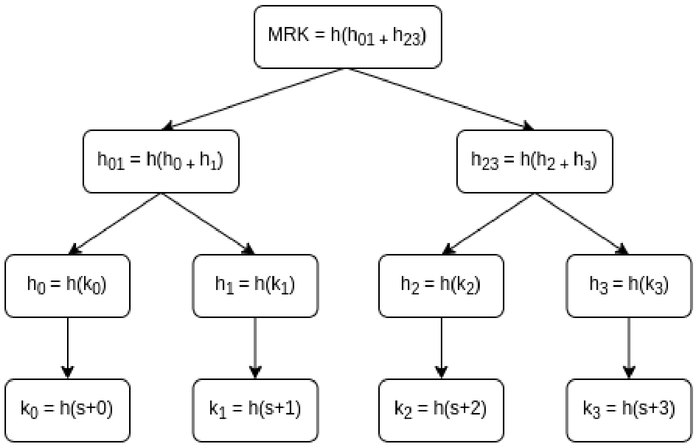

Figure 4: Tree of keys $MHT(K)$. In this example, MRK=H_{0123}.

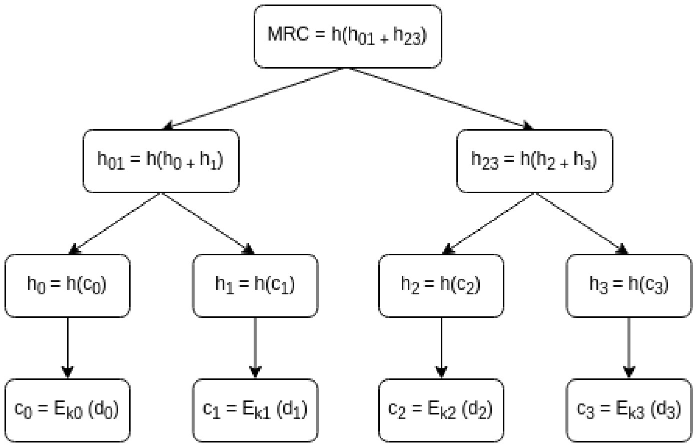

Figure 5: Tree of cryptograms $MHT(C)$. In this example, MRC=H_{0123}.

4.3.4 Protocol Execution

Once the protocol preparation phase is completed, the consumer requests the provider to reveal some sample portions. If these samples convince the consumer about the quality of the dataset, the consumer commits the payment. It is important to note that, if this protocol execution phase ends as expected, the protocol is completed, and there is no need to execute the conflict resolution procedure.

Figure 6 details the interactions between provider, consumer and smart contract during the execution phase of the DEFS protocol. The steps of this phase are enumerated as:

1. **Consumer**: Generate R=$[r_0 ... r_{v-1}]$ — $\forall r_i \in \{0 ... n-1\}$.

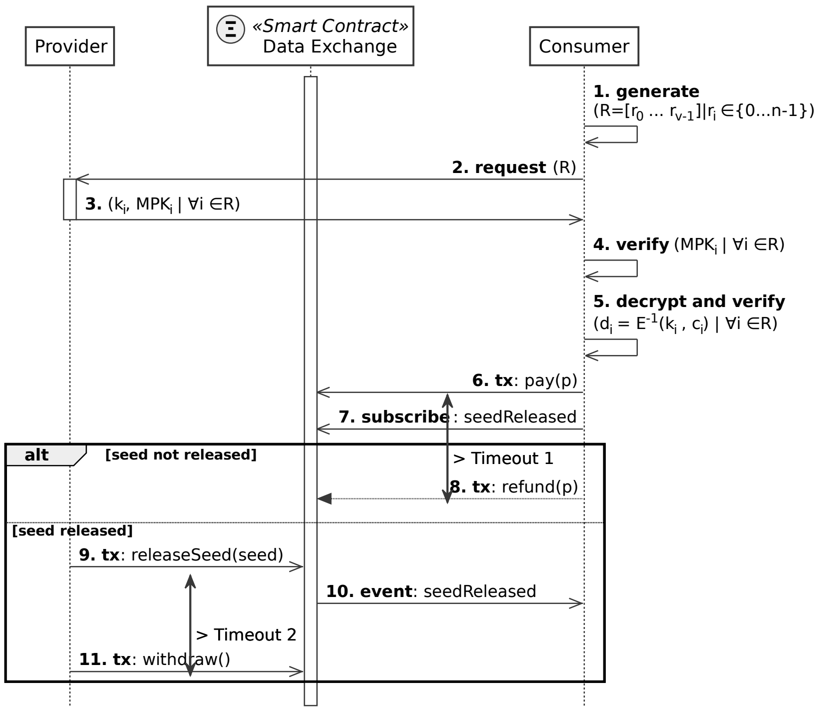

Figure 6: Protocol execution: sequence diagram.

The consumer selects at random the set of indexes R, which correspond to the sample portions to be revealed (for free). We consider that the provider should not decide on its own which data samples will be revealed, because she could decide to use a biased (not fair) set of samples. The consumer is also not able to choose on purpose particular registries, because previously the dataset was randomly sorted by the provider. Therefore, the consumer will choose at random an array of v values within the range 0 to n-1, corresponding to the indexes of the sample portions. As none of the entities control which registries are going to be disclosed, fairness in this process is assured.

2. **Consumer→Provider**: Request R which contains v indexes to be revealed.

   The consumer informs the provider of the v indexes of the sample portions to be revealed (R).

3. **Provider→Consumer**: Return k_i and MPK_i ∀i∈$\{R\}$.

   The provider discloses the keys associated to the v indexes, and the Merkle proofs to verify them in the $MHT(K)$. This process is done totally off-blockchain. Note that, allowing the consumer to choose v sample portions with no cost could result in attacks, as the consumer could try to get a large amount of free data by repeating the process of getting small samples. We do not consider this attack specially dangerous because the consumer cannot choose particular registries of the dataset. In addition, the provider can decide how many times she allows getting a sample set without making a final deal, and she can use the marketplace to blacklist abusive consumers. Nonetheless, a comprehensive analysis of potential attacks is made in Section 5.3. As shown in the analysis, the provider should carefully choose v and n to minimize the impact of such attacks.

4. **Consumer**: Verify each proof. MPK_i ∀i $\in \{R\}$.

   The customer should verify that the proofs sent by the provider match the MRK. To provide a partial example, consider the case of Figure 4, in which $n = 4$. Let us consider the case of one single sample portion $v = 1$, and the consumer has chosen index 3 to disclose. In this case, k_3 is sent to the consumer, and the proofs of the correctness are MPK_3=(h_{01},h_2). The consumer should verify if the following expression matches:

   $$hash(concat(h_{01}, hash(concat(h_2, hash(k_3))))) == MRK$$

   Notice that the expression is just calculating the root of $MHT(K)$, and comparing it with the MRK value published in the smart contract. If the values match, the key can be considered valid as it matches the proofs, and the consumer will continue with the protocol. If not, that means that the provider did not send the proper proofs and that the key is not verifiable. In this case, the consumer can abandon the protocol.

5. **Consumer**: Decrypt and verify each data sample.

   Now that the consumer is sure that the v keys are valid, he can decrypt the sample portions with the received keys:

   $$d_i = E_{k_i}^{-1}(c_i) \forall i \in R$$

   Once this is done, the consumer has to verify that the resulting data portions have the expected format:

   $$d_i = concat(i, data_i)$$

   If not, the data samples are invalid, and the consumer can end the protocol at this moment. If the format is valid, the consumer will evaluate the data samples $data_i$ ∀i $\in R$. If the samples do not convince the consumer to pay for the whole dataset, the protocol ends here, but if they do, he will continue with the following steps.

6. **Consumer→SC**: Transaction committing payment.

   If the samples convinced the consumer, he will send a transaction to the smart contract with the payment (p) to buy the dataset.

7. **Consumer→SC**: Subscription to seed revelation.

   The consumer also subscribes to the 'seedReleased' event, expecting to receive a notification when the provider publishes the seed to allow the complete data decryption.

8. **(Timeout 1) Consumer→SC**: Seed not released and refund to the consumer.

   If the provider has not released the seed in time, a first timeout ($Timeout1$) will expire and after that, the smart contract will allow the consumer to refund the payment. This is an unhappy path in the protocol.

9. **Provider→SC**: Transaction publishing the seed.

   The provider discloses the seed value s via a blockchain transaction before $Timeout1$ expires. This is the happy path of the protocol.

10. **SC→Consumer**: Event to consumer about seed revelation.

    As result of executing the transaction, the smart contract generates an event and the consumer will be notified that the seed value s has been revealed. The smart contract stops the Timeout 1 (due to seed revelation), and starts the Timeout 2 allowing the consumer to start a conflict resolution. Once that the consumer has the seed, it can derive all the keys:

    $$k_i = hash(s+i)\forall i \in \{0..(n-1)\}$$

    Now, the consumer has all the cryptographic material to decrypt the cryptograms C, but previously it has to verify that all the keys are correct. The procedure is similar to the one performed in Step 4, but with the difference that now the consumer has the capacity of re-generating the whole $MHT(K)$. If the consumer detects that (one or several) keys were not properly generated, he can start the optional phase of conflict resolution to obtain a refund. On the contrary, if all the keys were properly generated, the consumer can decrypt the previously received cryptograms C and access to the whole set of dataset:

    $$d_i = E_{k_i}^{-1}(c_i)\forall i \in \{0..(n-1)\}$$

    The consumer has also to verify that the d_i have the proper format, in the same manner that was done in Step 5, but for all the cryptograms. If the consumer detects that one or more cryptograms were not properly generated (they do not have the proper format), he can start the optional phase of conflict resolution to obtain a refund[1]. On the contrary, if all the cryptograms (once decrypted) have the proper format, the consumer has the complete dataset and can consider the protocol ended.

11. **(Timeout 2) Provider→SC**: Withdraw and protocol end.

    If Timeout 2 expires, it means that the consumer considers that the keys were properly generated, and cryptograms have the proper format (because if not, it would have previously started the conflict resolution). In this case, the provider can send a transaction to the smart contract to withdraw the payment, and end the protocol.

4.3.5 Conflict Resolution

The final aspect that our protocol has to solve is the conflict resolution, which may appear if the consumer detects a misbehaviour. As previously stated, the consumer must start the conflict resolution phase before the expiration of Timeout 2 in the protocol execution phase. If Timeout 2 expires, the smart contract considers the protocol ended and the provider can receive the payment.

We will consider two cases that are relevant and can end with a refund if the consumer is able to demonstrate the misbehaviour: (1) A key is not properly generated; and (2) A decryption of a cryptogram does not have the proper format. There are also a couple of extra cases for dispute that will not end with a refund: (3) Cryptograms not properly generated; and (4) Dataset of bad quality.

1. **Key not properly generated**: When consumers obtain the seed s in Step 9 of the protocol execution phase, they are able to generate the whole set of keys K. The way to check if the set of keys is compliant or not is by generating the whole set K with the formula $k_i = hash(s + i)$, and also re-constructing the whole $MHT(K)$ and verify that the root calculated matches the one published in the smart contract. At this moment, a consumer can know that the registered MRK is incorrect. However, in general, he cannot detect which keys were not properly generated. Although not possible in general, there are particular cases in which the consumer can do the detection of wrong keys. The detection is possible when the wrong keys are either one of the keys used for encrypting the samples, or a sibling key of them. In both cases, the consumer has got an MPK from the provider that matches the registered MRK. Then, if one of those keys does not follow the agreed format hash(i+s), the consumer can send the hash of the wrong key, its MPK, and the index in conflict to prove to the smart contract that the provider committed an incorrect MRK. For simplicity, we will assume that the consumer detects one single not-compliant key k_e, but this discussion is completely valid in case of

[1]Notice that, at this point, the consumer cannot argue that the cryptograms do not match the $MHT(C)$, because this should have been verified in the protocol preparation phase.

having multiple not compliant keys[2]. The following are the steps that the consumer, smart contract and provider have to follow during this conflict resolution about a specific key k_e. The sequence diagram associated is detailed in Figure 7:

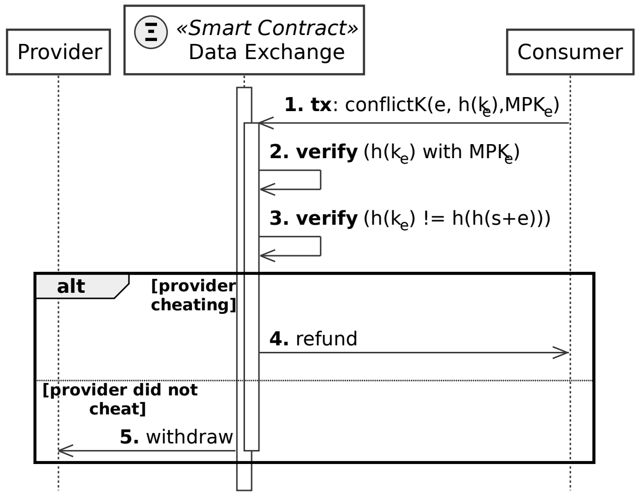

Figure 7: Protocol resolution-k: sequence diagram.

(a) **Consumer→SC**: tx: conflictK$(e, h(k_e), MPK_e)$
This conflict resolution is performed with a transaction from the consumer to the smart contract calling the function 'conflictK'. In this transaction, the consumer sends as parameters the problematic index (e), the hash of the invalid key $(h(k_e))$, and its associated proof (MPK_e).

(b) **SC**: Verify $h(k_e)$ with the MPK_e
The smart contract verifies that $h(k_e)$ and MPK_e match the MRK from the Step 9.

(c) **SC**: Verify $h(k_e)$!= $h(h(s + e))$
The smart contract verifies whether $h(k_e)$ matches $h(h(s+e))$ or not. If there is a match, it means that the provider did not cheat, while if the check does not match, then it means that the provider cheated.

(d) **SC**: Provider cheating
If the provider cheated, the smart contract refunds the consumer. In this case, the consumer will receive the price of the data (p) and also the cost of the transactions he sent. The cost of the transactions is taken from the provider's *collateral*.

[2]DEFS do not distinguish between one or several not compliant keys, in case of demonstrating that one single key is not properly generated, all the payment will be refunded to the consumer.

(e) **SC**: Provider not cheating

If the provider did not cheat, the smart contract automatically transfers the payment (p) to the provider.

2. **Cryptograms do not have the proper format**:

This situation happens when there is a conflict in D, and so one or several data portions do not have the proper format. Just remark that there was a previous checking of this type in Step 5 of the protocol execution phase, in which v of the possible sample portions were tested to see if they had the proper format:

$$d_i = concat(i, data_i)$$

However, not all the data portions were tested (only v of n). For simplicity, we will assume that there is one single not-compliant portion d_e, but this discussion is completely valid in case of having multiple not compliant portions.

This scenario implies that the decryption of a c_e results in a d_e that does not correspond with the expected format d_e=concat(e,$data_e$). Specifically, the decrypted cryptogram does not start with the expected index (e). In this case, the consumer can start the conflict resolution about the format of data. Figure 8 shows the sequence diagram about this scenario.

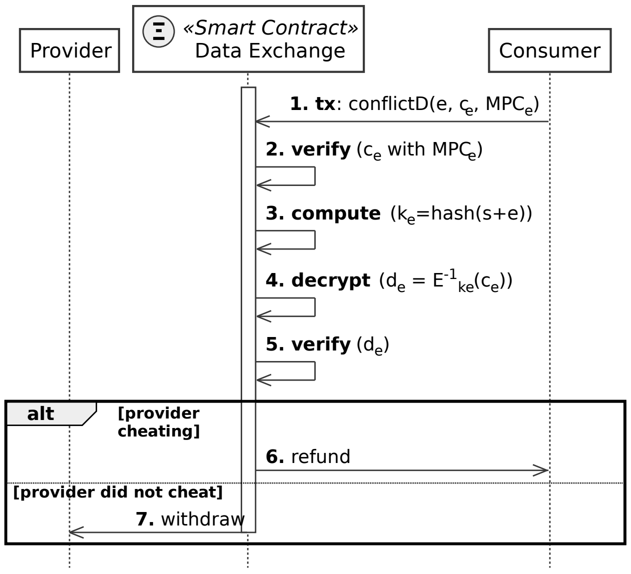
Provider
«Smart Contract» Data Exchange
Consumer
1. tx: conflictD(e, c_e, MPC_e)
2. verify (c_e with MPC_e)
3. compute (k_e=hash(s+e))
4. decrypt (d_e = E^-1_ke(c_e))
5. verify (d_e)
alt
[provider cheating]
6. refund
[provider did not cheat]
7. withdraw

Figure 8: Protocol resolution-d: sequence diagram.

(a) **tx: conflictD**:

The conflict resolution about a c_e starts with a transaction from the consumer to the smart contract. To do that, the transaction calls the smart contract function 'conflictD' and sends as parameters the problematic index (e), the cryptogram involved c_e, and the proofs of the validity of this particular cryptogram. The intention of the consumer is to show that the cryptogram c_e was properly generated by the provider, but that after decrypting it with k_e, the resulting data portion has a bad format.

In this case, the protocol can be resolved in a single transaction from the consumer, because the smart contract can compute k_e. Note that there is no need for an extra timeout, because the provider does not need to send anything, all the proofs are available for the smart contract to compute.

(b) **Verify** MPC_e:

The smart contract has to verify that the cryptogram provided by the consumer is valid, that is to say, that calculating MRC for the problematic cryptogram c_e from MPC_e is coherent with the root stored in the smart contract. To provide some piece of example, consider the case of Figure 5, in which $n = 4$. Let us consider the case of demonstrating that c_3 is properly generated. In this case, the proofs of the correctness of c_3 are MPK_3=(h_{01},h_2). The smart contract should verify if this expression matches:

$$hash(concat(h_{01}, hash(concat(h_2, hash(c_3))))) == MRC$$

Notice that the expression is just calculating the root of $MHT(C)$, and comparing it with the MRC value published in the smart contract. If these values match, the cryptogram can be considered valid as it matches the proofs, and the protocol continues with the following step. If not, the consumer did not send the proper proofs to demonstrate that the provider was cheating, that automatically receives the withdrawal of the money as shown in Step 2g.

(c) **Compute** k_e:

Now, the smart contract knows that the cryptogram c_e is valid. Next, the smart contract computes the associated key $k_e = hash(s + e)$. Remember that s was published by means of the transaction sent in Step 9 of the protocol execution phase.

(d) **Decrypt** c_e:

The smart contract has the valid key and the valid cryptogram, so it is able to decrypt and obtain d_e=$E^{-1}_{k_e}(c_e)$.

(e) **Verify** d_e **format**:

The smart contract can verify if the data portion d_e has the correct format.

(f) **refund**:

If the data portion d_e does not start with the index e, the provider was cheating, so the transaction ends transferring the costs (p+$collateral$) to the consumer.

(g) **withdraw**:

If the data portion d_e start with the index e, the provider was not cheating, and the transaction ends transferring the payment (p) and the *collateral* to the provider.

3. **Cryptograms not properly generated**

   This case happens when there is a conflict in C, and one or several cryptograms do not match the root MRC of the tree $MHT(C)$. As previously stated, the consumer receives all the cryptograms C in Step 10 of the protocol preparation, and verifies the correctness of the whole set of cryptograms in Step 13. The consumer was responsible for verifying the $MHT(C)$ at this very moment, and if any problem during this checking was found just abort the protocol before committing any payment. But if the consumer continued with the protocol, he was tacitly accepting the correctness of the generation of the cryptograms and the corresponding $MHT(C)$ and MRC. In case the consumer detects a cryptogram (c_e) not matching the MRC at the protocol execution or protocol resolution phases, he cannot try to get a refund, and for this reason the conflict resolution is not considering that case.

4. **Dataset of bad quality**:

   This case happens when the consumer obtains a valid data portion d_e (with the correct format $d_e = concat(e, data_e)$, but the content has not the quality expected by the consumer. In this particular case, the DEFS protocol is not able to consider the quality of the dataset[3], so this case is out of its scope of the discussion and no refund can be asked from the consumer's side. The v sample portions that were disclosed in Step 5 of the protocol execution (for free, prior to any payment) try to alleviate this possibility. In any case, the marketplace can consider to have a reputation tool to value data providers and try to avoid this kind of behaviours.

4.3.6 State Diagram

The protocol operation and the interactions between the different stakeholders and the smart contract are detailed in Figure 9.

[3]Probably, assessing the goodness of a dataset requires human interaction and cannot be made automatically by the smart contract.

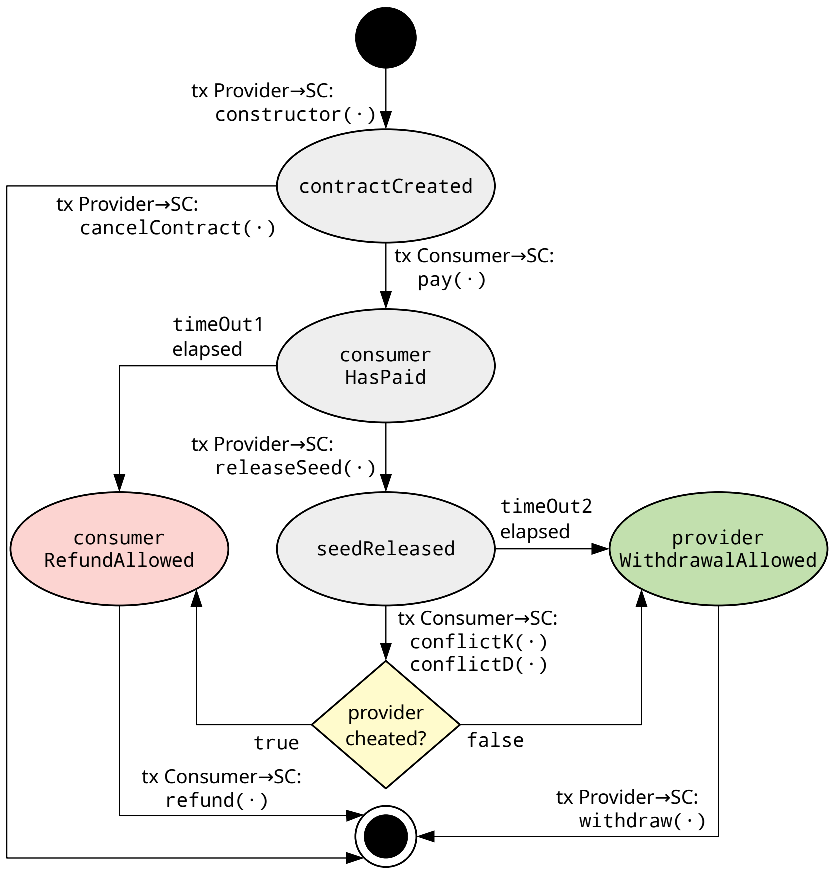

Figure 9: State diagram of the smart contract.

5 Security Analysis

DEFS enables providers and consumers to make commercial agreements via a smart contract. Essentially, the smart contract ensures that, if a consumer agrees to purchase a dataset based on the provided random samples, then the provider receives the right amount of money and the consumer gets access to the whole dataset.

5.1 Protection Against Channel Attacks

Channel attacks are those in which the attacker purposely tries to eavesdrop information from the channel. In our proposal two different channels must be considered: the on-chain channel with the interactions with the smart contract, and the off-chain channel between the data provider and the data consumer.

It is assumed in this work that the off-chain channel is protected from overhearing and tampering, for instance by forcing it to use transport layer security (TLS). As a result, no information would be exposed in this channel. However, the on-chain channel is public and all the interactions with the smart contract will be available for an attacker, namely the data-exchange identifier id, number of data portions n, the price of the dataset p, and the roots of the MHT of cryptograms MRC and keys MRK.

The only valuable information an attacker could obtain is the size n and price p of the data set. The data-exchange identifier id is just an identifier. The MRC and MRK leak no information regarding the keys and cryptograms, since the cryptographic hash function used to create the MHT is assumed to be preimage- and collision-resistant.

5.2 Consumer's Protection Against Provider Attacks

In the first part of the protocol, the consumer gets access to a set of free random samples from the dataset. The provider commits the structure of cryptograms and keys structure (when committing the MRC and MRK to the smart contract). Then, the consumer can request a specific set of random samples before making an assessment with regard to buying the dataset or not.

A malicious provider would like to send a selected set of samples that make the dataset more appealing. However, the provider has no control over the selected samples and changing them on the fly would require the fake sample keys and cryptograms to collude with the committed ones. This attack is assumed to be unfeasible since the probability of finding collisions in the cryptographic hash function used to generate the Merkle trees is assumed to be negligible.

Providers would also hold datasets with both good and bad data, meaning bad fake or even duplicated data. However, since providers cannot choose the requested free samples, bad samples could be detected during the evaluation of the free samples. In any case, it is up to the consumer to decide if the amount of samples is representative enough to get a fair idea of the content.

Finally, another potential attack would be that of a provider releasing wrong keys, wrong cryptograms or incorrectly encrypted data after getting paid. In these cases, the consumer can make use of the different conflict resolutions explained in section 4.3.5 and get a refund. Recall that, in the case of incorrect cryptograms, it is important that the consumer validates all cryptograms before doing the payment, since otherwise he will not have access to the conflict resolution.

5.3 Provider's Protection Against Consumer Attacks

Every time a consumer engages in the protocol, he receives v samples of the product. A dishonest consumer with several identities could accumulate free samples and try to get the whole dataset without paying for it or, equivalently, collaborate with other consumers to get as many data samples as possible for free. As we show hereunder, the provider can adjust the amount of samples disclosed to the consumer to reduce the probability of these attacks succeeding.

Before analysing a general setup, let us consider a simple example in which the provider has 2 different samples and every consumer gets 1 for free. That is, $n = 2$ and $v = 1$. Let us compute what is the probability that a malicious consumer with several identities gets both samples. With one interaction the consumer gets only half of the product for free. If the consumer creates a new identity, he would get a new sample, but he would only get the whole product if the new sample is different from the previous one. So, we need to compute what is the probability that a new sample is different from the previous one. We can think of this scenario as tossing two coins and finding the probability that we get a head (H) and a tail (T), which is 1/2. So, with two identities, a dishonest consumer would have 50% chances to get the whole set of samples. If the consumer creates a new identity and gets a third sample, the probability of him getting the two different samples goes up to 75%, which is the probability that in a sequence of 3 coin tosses, at least one is a head and another a tail.

In general, assume that a consumer gets v random samples and is able to interact with the provider with k different identities, obtaining a total of $m = k \times v$ free random samples. Like before, the probability that among m samples, n of them are different, is the same as the probability of having n different elements in a sequence of m elements. If $m < n$, then clearly this probability is 0. If $m = n$, then there are $n!$ different sequences with the n elements. So, the probability of getting a sequence of this kind is $n!/n^n$. When n is large enough, this probability is very low. To get an idea, we show in Table 2 what happens if the provider discloses 10% of samples of his dataset and the consumer creates 10 identities to get a total of n samples. Note that, even for small values of n, the probability that a consumer gets the whole dataset is very low.

n	v	$k = m/v$	Probability of getting the whole dataset
10	1	10	3.62×10^{-4}
100	10	10	9.33×10^{-43}
1000	100	10	4.02×10^{-433}
10.000	1.000	10	$\sim 10^{-4340}$
100.000	10.000	10	$\sim 10^{-43426}$

Table 2: Probability that the consumer gets the whole dataset if the provider makes 10 interactions with the consumer and discloses 10% of his dataset every time.

To increase the odds of getting all samples, a dishonest consumer would create more identities so that $m \geq n$. In this scenario, we need to calculate what is the probability that, among the n^m possible outcomes, the consumer gets n distinct data samples. If we go back to the case $n = 2$, $v = 1$, and a consumer with three identities ($k = 3$), we should count how many sequences of three elements contain 2 distinct elements. Or equivalently, what is the probability that after three coin tosses, we get at least one head and one tail. We can think of this problem as counting the different ways in which we can assign the three positions of the sequence $\{1, 2, 3\}$ to a head or a tail, so that at least one is a head and another a tail. This counting is precisely the number of ways in which we can partition the set $\{1, 2, 3\}$ into two non-empty sets: $\{1, 2\} \cup \{3\}$, $\{1, 3\} \cup \{2\}$ and $\{2, 3\} \cup \{1\}$. If we assign the first set to heads and the second set to tails, the partitions lead to the three sequences HHT, HTH, TTH, and if we do the opposite assignment, we get TTH, THT, HHT. As a result, we get a total of $3 \times 2 = 6$ different sequences containing at least one head and one tail, which divided by the $2^3 = 8$ possibilities, results in the 75% chances we claimed before.

In general, the number of sequences of m elements that contain n distinct elements, is equivalent to the number of ways we can partition the m positions of the sequence into n non-empty sets multiplied by the number of permutations of n distinct elements. This count is precisely the Stirling number of the second kind $S(m, n)$ multiplied by the number of permutations of n elements [26, Ch. 3]:

$$S(m, n)n! = \sum_{j=0}^{n} \binom{n}{j} (-1)^{n-j} j^m.$$

Hence, the probability that a consumer with $k = m/v$ identities gets the whole data set is

$$\frac{S(m, n)n!}{n^m} = \frac{1}{n^m} \sum_{j=0}^{n} \binom{n}{j} (-1)^{n-j} j^m.$$

In Figure 10 we illustrate the probability that a consumer with k identities that gets 10% of free samples with each identity, obtains the whole data set of n registries. Note that a provider with a data set of 10,000 registries disclosing a random 10% of it for free should almost only be worried about consumers that are able to create more than 50 different identities. The off-chain identification system (see Section 4.3.1) should be chosen to minimize or even impede the likelihood of an attacker getting more than k identities. Typically, n is several orders of magnitude higher than 10,000, so the amount of fake identities needed to perform this attack would make it infeasible in practice.

Even if the probability of getting the whole dataset is low enough, the provider may also want to avoid disclosing a large valuable set of data. We analyse what is the probability of obtaining a meaningful amount of samples for different values of m. The same way as we argued before, the probability of getting x different samples is the number of combinations of sequences of m elements with x distinct elements, but now multiplied by the combinations of x elements that we can make with n distinct elements. That is,

$$P(\text{get exactly } x \text{ distinct samples}) = \frac{S(m, x)x!}{n^m} \binom{n}{x} = \frac{1}{n^m} \binom{n}{x} \sum_{j=0}^{x} \binom{x}{j} (-1)^{x-j} j^m.$$

Therefore, the probability of getting x distinct elements in a sequence of m

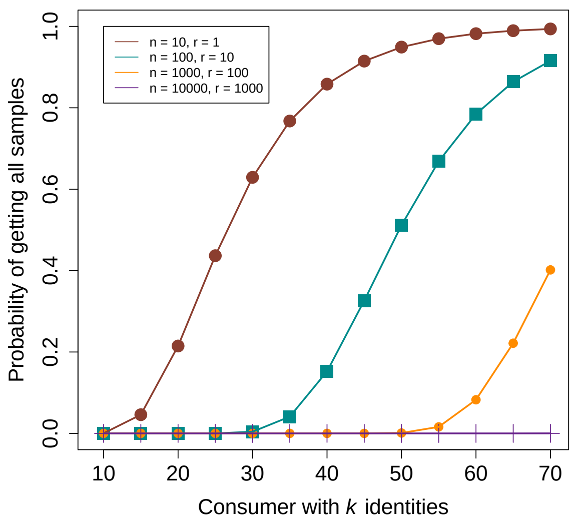

Figure 10: Probability that a consumer with k different identities that gets a 10% of samples with each identity obtains all n different data samples.

elements is

$$\begin{aligned} P(\text{get at least } x \text{ distinct samples}) &= 1 - \sum_{i=0}^{x-1} P(\text{get exactly } i \text{ distinct samples}) \\ &= 1 - \sum_{i=0}^{x-1} \binom{n}{i} \frac{S(m,i)i!}{n^m}. \end{aligned}$$

To illustrate the tendency of these probabilities, we have depicted in Figure 11 the probability of obtaining at least $5, 10, 15, \ldots, 100$ different samples from a dataset with 100 different samples. The different lines correspond to the probabilities using a different number of identities.

As we can see, there is a significant drop in the probability of obtaining at least a 65% of samples with 10 identities, but with more identities, this inflection point moves up to 80% (with $k = 15$), 90% (with $k = 20$), and 95% (with $k = 25$). So, even though the probability of obtaining the whole 100% of the data set is very close to 0, with 25 identities it is possible to obtain at least the 90% of it with probability 0.84.

As we have shown, the provider can mitigate the risks of identity-replication by strongly authenticating consumers through the off-chain channel (see Section 4.3.1), and by adjusting the amount of free samples disclosed to the consumer.

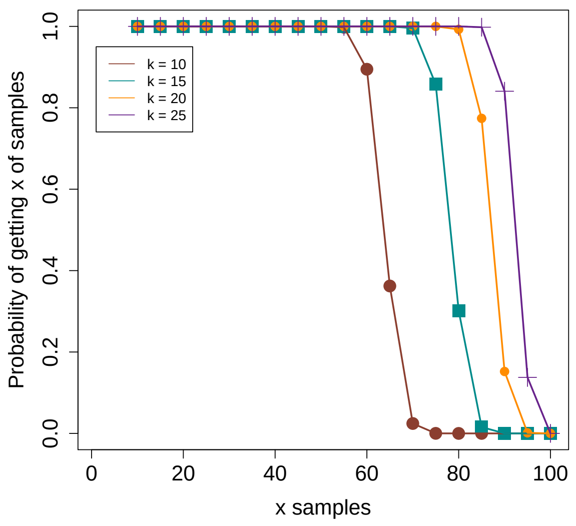

Figure 11: In a database of $n = 100$ portions and $v = 10$ free samples per consumer, this graphic depicts the probability of getting a different portion of samples for free with several identities: $k = 10$ (large red circles), $k = 15$ (blue squares), $k = 20$ (small orange circles), $k = 25$ (purple lines).

In general, the provider should find a trade-off between securing the dataset while at the same time letting the consumer get a fair idea its content.

Lastly, we would like to remark that the provider is protected against a consumer that does not pay after acknowledging the protocol, since the salt is revealed only once the payment is in the smart contract, and the cryptograms are useless without it.

6 Conclusions

Distrust is one of the main obstacles to implement exchanges between data providers and data consumers in a decentralized way. In this article, we present a protocol that allows a consumer to probabilistically obtain and check a subset of a dataset on sale from a provider before committing the payment. The protocol is executed using a smart contract deployed in a public distributed ledger. Once the consumer accepts to buy the dataset, the payment process, the agreed terms, and the possible refunds are managed and enforced by the smart contract. To expose the dataset, our protocol splits the data in portions and encrypts and stores each portion off-chain. Then, we create a MHT for the cryptograms and another MHT for the encryption keys. The encryption keys are related to each other using a cryptographic hash function in a way that allows us to implement a cost-efficient conflict resolution mechanism. The security analysis of our protocol shows that consumers and providers are economically protected and that the provider can reduce the risks of identity-replication attacks by adjusting the amount of free samples disclosed to the consumer.

Acknowledgements

The architecture presented in this paper is supported and developed in the context of the i3-MARKET project [**?**]. The i3-MARKET project is an active European H2020 project focused on developing solutions for building an European data market economy by enhancing current marketplace platforms with innovative technologies (call H2020-ICT-2019-2 with grant agreement number 871754). This work is also supported by the TCO-RISEBLOCK (PID2019-110224RB-I00), MINECO/FEDER funded project ARPASAT TEC2015-70197-R and by the Generalitat de Catalunya grant 2014-SGR-1504.

References

[1] V. Gopalkrishnan, D. Steier, H. Lewis, and J. Guszcza, "Big data, big business: Bridging the gap," in *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, ser. BigMine '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 7–11. [Online]. Available: https://doi.org/10.1145/2351316.2351318

[2] L. D. W. Thomas and A. Leiponen, "Big data commercialization," *IEEE Engineering Management Review*, vol. 44, no. 2, pp. 74–90, Second 2016.

[3] N. Ravi and N. R. Sunitha, "Introduction of blockchain to mitigate the trusted third party auditing for cloud security: An overview," in *2017 2nd International Conference On Emerging Computation and Information Technologies (ICECIT)*, 2017, pp. 1–6.

[4] I. Bashir, *Mastering blockchain.* Packt Publishing Ltd, 2017.

[5] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "Blockchain technology overview," https://arxiv.org/abs/1906.11078, 2019, [Accessed 10-May-2021].

[6] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project White Paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[7] F. Haider, "Compact sparse merkle trees," Cryptology ePrint Archive, Report 2018/955, 2018, https://eprint.iacr.org/2018/955.

[8] R. Dahlberg, T. Pulls, and R. Peeters, "Efficient sparse merkle trees: Caching strategies and secure (non-)membership proofs," Cryptology ePrint Archive, Report 2016/683, 2016, https://eprint.iacr.org/2016/683.

[9] S. Al-Kuwari, J. H. Davenport, and R. J. Bradford, "Cryptographic hash functions: Recent design trends and security notions," Cryptology ePrint Archive, Report 2011/565, 2011, https://eprint.iacr.org/2011/565.

[10] H. Yoo and N. Ko, "Blockchain based data marketplace system," in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, 2020, pp. 1255–1257.

[11] L. Mikkelsen, K. Mortensen, H. Rasmussen, H.-P. Schwefel, and T. Madsen, "Realization and evaluation of marketplace functionalities using ethereum blockchain," in *2018 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC)*, 2018, pp. 47–52.

[12] V. P. Ranganthan, R. Dantu, A. Paul, P. Mears, and K. Morozov, "A decentralized marketplace application on the ethereum blockchain," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, 2018, pp. 90–97.

[13] A. Braud, G. Fromentoux, B. Radier, and O. Le Grand, "The road to european digital sovereignty with gaia-x and idsa," *IEEE Network*, vol. 35, no. 2, pp. 4–5, 2021.

[14] K. R. Özyilmaz, M. Doğan, and A. Yurdakul, "Idmob: Iot data marketplace on blockchain," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2018, pp. 11–19.

[15] D.-D. Nguyen and M. I. Ali, "Enabling on-demand decentralized iot collectability marketplace using blockchain and crowdsensing," in *2019 Global IoT Summit (GIoTS)*, 2019, pp. 1–6.

[16] P. Tzianos, G. Pipelidis, and N. Tsiamitros, "Hermes: An open and transparent marketplace for iot sensor data over distributed ledgers," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, pp. 167–170.

[17] V. Arya, S. Sen, and P. Kodeswaran, "Blockchain enabled trustless api marketplace," in *2020 International Conference on COMmunication Systems NETworkS (COMSNETS)*, 2020, pp. 731–735.

[18] S. Musso, G. Perboli, M. Rosano, and A. Manfredi, "A decentralized marketplace for m2m economy for smart cities," in *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2019, pp. 27–30.

[19] G. S. Ramachandran, R. Radhakrishnan, and B. Krishnamachari, "Towards a decentralized data marketplace for smart cities," in *2018 IEEE International Smart Cities Conference (ISC2)*, 2018, pp. 1–8.

[20] B.-G. Jeong, T.-Y. Youn, N.-S. Jho, and S. U. Shin, "Blockchain-based data sharing and trading model for the connected car," *Sensors*, vol. 20, no. 11, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/11/3141

[21] Y.-N. Li, X. Feng, J. Xie, H. Feng, Z. Guan, and Q. Wu, "A decentralized and secure blockchain platform for open fair data trading," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 7, p. e5578, 2020, e5578 cpe.5578. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5578

[22] S. Ma, Y. Mu, and W. Susilo, "A generic scheme of plaintext-checkable database encryption," *Information Sciences*, vol. 429, pp. 88–101, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025517301640

[23] W. Dai, C. Dai, K.-K. R. Choo, C. Cui, D. Zou, and H. Jin, "Sdte: A secure blockchain-based data trading ecosystem," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 725–737, 2020.

[24] P. standardization initiativ, "Json for linking data," https://json-ld.org/, 2021.

[25] J. Muñoz, J. Forne, and O. Esparza, "Certificate revocation system implementation based on the merkle hash tree," *IJIS*, vol. 2, p. 110–124, 2004.

[26] T. Mansour and M. Schork, *Commutation Relations, Normal Ordering, and Stirling Numbers*, ser. Discrete Mathematics and Its Applications. Chapman and Hall/CRC, 2015.

Rafael Genés-Durán is currently a PhD candidate of the Information Security Group (ISG) doing research in distributed ledger technologies and zero-knowledge proofs at Universitat Politècnica de Catalunya. He holds a B.S. degree in Telecommunications Engineering (2017) and a Master in Informatics Engineering (2019). Contact him at `rafael.genes@upc.edu`.

Juan Hernández-Serrano is an associate professor of the Department of Network Engineering of the Universitat Politècnica de Catalunya in Spain, and a researcher of the Information Security Group (ISG). He holds an M.S. in Network Engineering (2002) and a Ph.D. in the field of information security (2008). His research interests has been focused on different aspects of networks security and privacy, including IoT, distributed ledgers, cognitive radio networks, M2M, smart grids, eVoting and digital forensics. Contact him at `j.hernandez@upc.edu`.

Oscar Esparza is working as associate professor of the Department of Network Engineering of the Universitat Politècnica de Catalunya. He holds an M.S. in Telecommunications Engineering (1999) and a PhD in Security Engineering (2004). His expertise areas are related to network security and applied cryptography. Since 2017 he leads the Information Security Group (ISG) of the UPC. Contact him at `oscar.esparza@upc.edu`.

Marta Bellés-Muñoz received her B.S. degree in Mathematics at Universitat Autonoma de Barcelona and continued her Master studies at Aarhus University, where she focused on the study of elliptic curves and isogeny-based cryptography. She is currently a PhD student doing research on security and efficiency of arithmetic circuits for zero-knowledge proofs at Universitat Pompeu Fabra in collaboration with Dusk Network. Contact her at `marta.belles@upf.edu`.

Jose L. Muñoz-Tapia is a researcher of the Information Security Group (ISG) and an associate professor of the Department of Network Engineering of the Universitat Politècnica de Catalunya. He holds an M.S. in Telecommunications Engineering (1999) and a PhD in Security Engineering (2003). He has worked in applied cryptography, network security and game theory models applied to networks and simulators. His research focus has now tuned to distributed ledgers technologies, and he is the director of the Master program in Blockchain technologies at UPC School. Contact him at `jose.luis.munoz@upc.edu`.