Ledgys Distributed Network
Decentralized data markets with Ethereum
Draft 1.2

Sébastien Jehan, Gautier Marin, Quentin de Beauchesne
(contact@ledgys.io)

Contributor: Régis Gourdel

Advisor:
David Pointcheval (david.pointcheval@ens.fr)

September 26, 2016

Abstract

Ledgys Distributed Network is a peer-to-peer platform built on Ethereum enabling decentralized data markets. Blockchain technology is praised for its ability to ensure trustworthy transactions in a decentralized manner. The applicability of blockchains on fields other than transactions is growing, but the growth has been slower than expected mostly due to the structure inherited from its predecessors. The current blockchain as most know it has some pitfalls ranging from the inability to store large amounts of data to the negligence of data privacy. The storage abilities in the current systems are often limited to publishing and timestamping hashes of a dataset and then having the client retrieve the actual dataset elsewhere. The client can then verify the existence and the validity of the dataset against the hash stored on the blockchain.

Such data existence proofs are used by systems such as Sia and Storj, therefore enabling decentralized file storage by offloading the actual data storage offchain. Ledgys Distributed Network provides a platform to store and sell private structured data (that can be added in a database) in a decentralized way, with custom applications, using an open blockchain network such as Ethereum.

The network will be controlled by a distributed organisation, the Ledgys Network Distributed Organisation, which will perform due-diligence control on the applications, the smart contracts code and parameters. We will discuss why this new layer is necessary, how our network and organisation is designed and what problems it might encounter. We've also included a few examples on possible applications that can run on the Ledgys network.

Keywords: Blockchain, Ethereum, Distributed Database, Data Marketplace

1 Introduction

The Blockchain generated a huge interest from a variety of industries for its immutability and auditability. However, many businesses firmly believe that their sensitive data should remain local, away from prying eyes, and therefore chose to run a private blockchain or a distributed ledger database. A private blockchain is a blockchain that operates without any open consensus.

Ledgys knows that Blockchain technology is the best way to go, not only because it creates a global transactional backbone among companies, something which is otherwise hard and extremely costly to build and maintain, but also because it can trigger completely new business models. Without the open consensus, the blockchain protocol is much less challenged, thus Ledgys believes that the Blockchain should remain open to foster innovation and improve security ("given enough eyeballs, all bugs are shallow" [1]).

However, current Blockchain technologies lack affordable, distributed and secure data storage capabilities, and furthermore don't have any data privacy features. That's why Ledgys has proposed to build a private layer on top of the public blockchain Ethereum [2]. It will be used both for its classic Blockchain features (certification of data integrity with hashes stored on the blockchain), and for its "Smart contracts" transactions-based code, the backbone of a decentralized, autonomous, fully distributed, encrypted and secure network.

Retrieving historical and spot business data and past transaction history through current blockchain infrastructures is dreadful: there is no simple public API on the blockchain client to query smart contract data with useful parameters. The LDN defined a query language (similar to the SQL language but with extensions for defining data prices) to query data on distributed data nodes, allowing for fast and intuitive data retrieval of business data and past transactions. This query language is called the LQL (Ledgys Query Language).

Overall, the LDN infrastructure completes the Blockchain infrastructure to create a collaborative, large scale, distributed, transactional and secure database.

2 Ledgys Distributed Network Overview

Ledgys Distributed Network is a data market where data sellers and data buyers meet to exchange all kinds of data. The data sellers offer their encrypted data on the LDN and are paid by the data buyer in cryptocurrency. Each data Marketplace is structured by an application that defines the economical model and the data schema. The application role is to organize the match between data buyers and sellers through a web interface although the inherent design of transactions is peer to peer.

Generally, the application will also match data buyers and data sellers

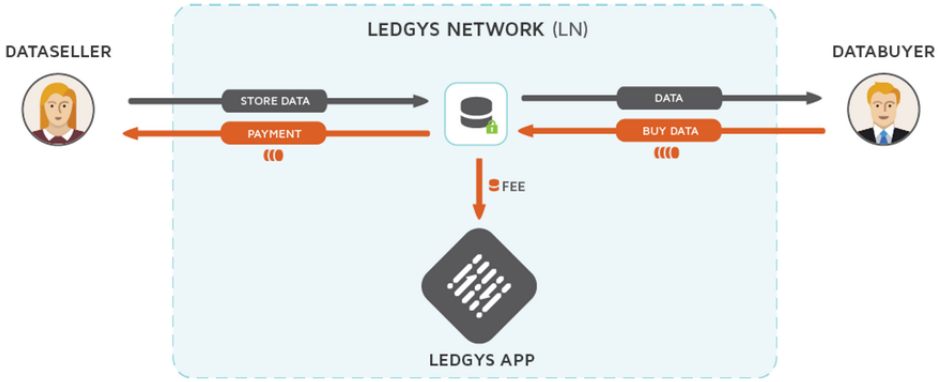

Figure 1: Data exchange overview on Ledgys Distributed Network

This exchange is made possible in a distributed environment by the following technologies:

- Distributed data storage: The distributed data storage infrastructure is supported by independent datanodes who sell their storage abilities to store data from various data sellers.
- Blockchain Contracts: The LDN intelligence is decentralized through the power of the Ethereum blockchain (application).
- Cryptography: Sellable data is stored encrypted in the datanodes. Independent decryption keys are transferred item by item
- Cryptocurrency: The network uses its own cryptocurrency for both storage and payments.

2.1 Datanodes Overview

We define a datanode as a computer that rents its data storage capacity to the Ledgys network. Datanodes are not required to be trusted entities on the Ledgys Distributed Network, as their good behavior is assured via smart contracts.

Datanodes are servers connected to Ledgys Distributed Network to provide hosting services. They store encrypted data from many data sellers. The data is fully replicated on several nodes, the storage of the data is periodically assessed through a distributed protocol (Proof of Storage). When the data buyer pays the data seller for the data, he receives the data directly from the Datanodes, thanks to a distributed intelligence.

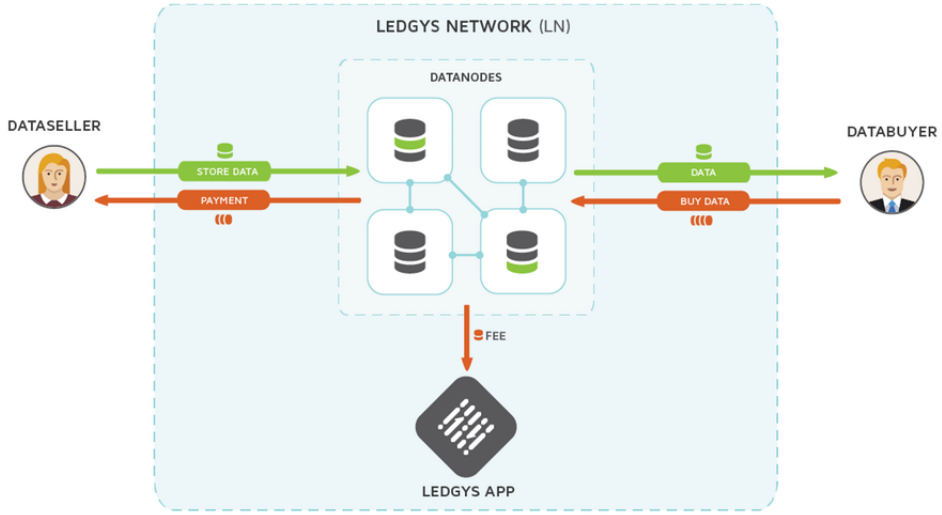

Figure 2: Datanodes in action

Detailed explanations on datanodes structure will be presented in part II, data storage management in part III and data exchanges in part IV.

2.2 Smart Contracts Overview

Ledgys Distributed Network is powered by the Ethereum Blockchain, which enables data integrity with proof of storage mechanisms. The distributed process among peers is orchestrated in smart contracts. The LDN will contain several smart contracts: the datanode management smart contract, the application management smart contract, storage contracts for datasellers and application contracts for data purchases.

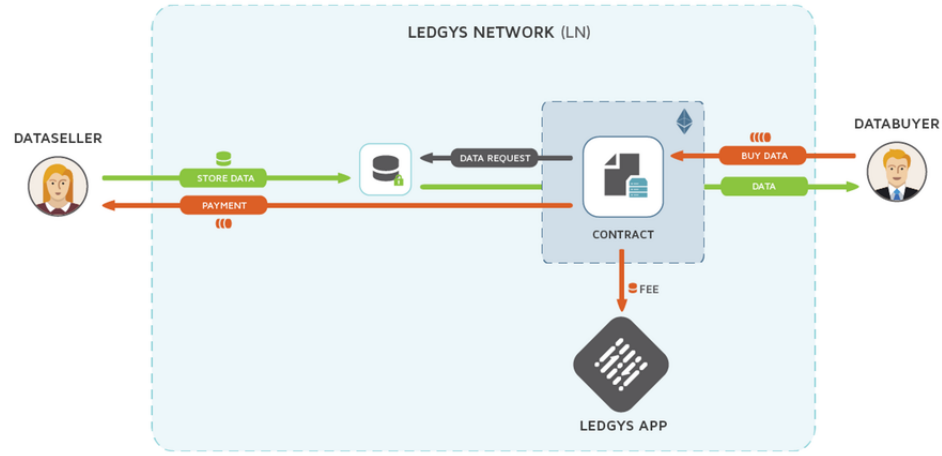

Figure 3: Smart contracts overview

In Ledgys Distributed Network, databuyers perform queries to get encrypted data (data request) and pay some fees to a smart contract to receive decryption keys

(called "decryption pieces") for the requested data. The smart contract pays the dataseller for the requested data, minus an application fee split between the application owner and the LNDO (Ledgys Network Distributed Organisation), an organization on top the Ledgys Distributed Network (detailed in part VI).

2.3 Cryptography

DataNodes store segments with encrypted data to sell and clear data to perform SQL queries. The data is encrypted using an algorithm where linear computation can be performed on column elements, and the decryption keys are sent only for requested items, not for the whole database. Each decryption key is shared among several key holders.

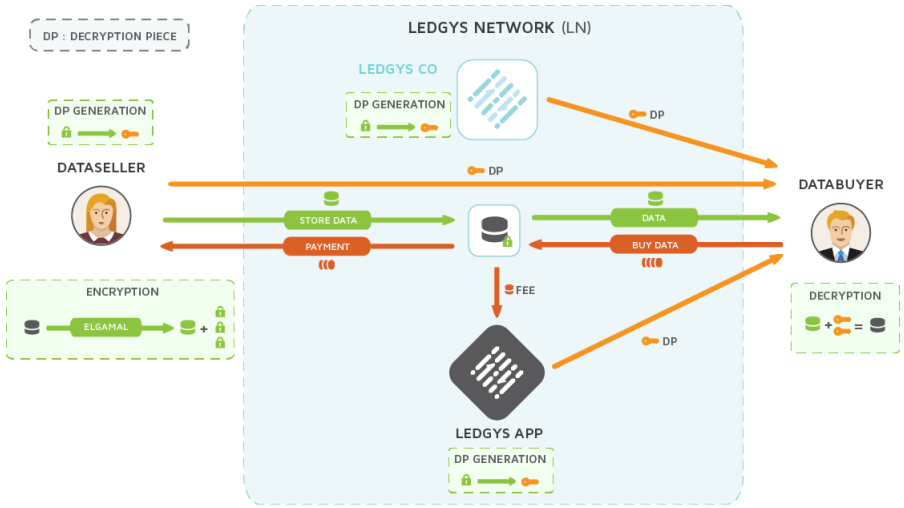

Figure 4: Decryption pieces distribution on Ledgys Distributed Network

When the databuyer receives the encrypted data, it also receives the keys to decypher it from various key holders. He then can use the decrypted data locally (this part is explored in part V).

2.4 Cryptocurrency

Ledgys Distributed Network uses its own currency (labelled XLC) both for both data and storage payments. For each purchase, the dataseller will receive the payment price minus a fixed fee for the application associated to it, and a dynamic fee associated to the LNDO (a discount is applied depending on the transaction volume of the application).

The LNDO share holders (holding XLS tokens) will collect their fees each month.

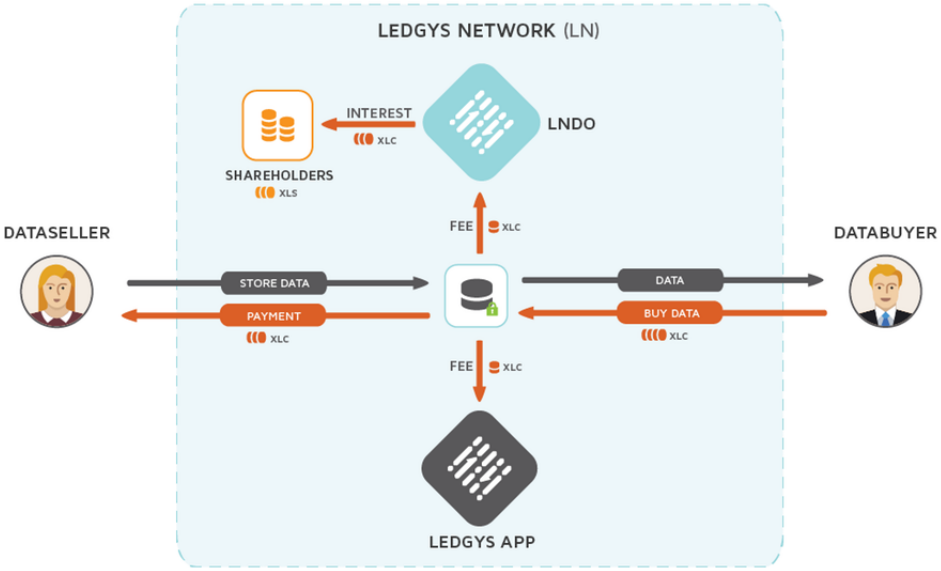

Figure 5: Payments and fee sharings on Ledgys Network

We will explore the economy of the network in part VII.

3 Datanodes structures

3.1 Data Structure

App owners define the common datatable structure shared by data sellers. They play a structuration role for their registered data seller nodes, thus defining column names and types, whether columns are encrypted and whether computation is possible on each column.

App owner also define technical hosting parameters, such as datatables segment size for all datasellers, a parameter that defines the constraints of the database integrity control (a proof that the data is kept unchanged when downloaded from a datanode). The segment size is a standard byte-multiple going from 2 MB to 2048MB. He also specifies the minimum redundancy level, an integer defining how many times the data will be duplicated on the datanodes.

Datasellers publish their database in accordance with the app owner datatables schemas. Each sellable data cell is encrypted using our custom encryption scheme, with custom embedded behaviors if some computation must be performed by the databuyer on the encrypted data. Each datatable is then split into uniform segments, whose sizes have previously been defined by the app owner. Each segment contain extra technical information that defines the table, the database they originate from, the associated application owner, the encryption parameters, as well as the price of the data (pricing schemes are defined later).

Additionally, any sellable data will be encoded as many times as the dataseller chosen redundancy level, so that no datanode could claim to store multiple copies of

the data while actually storing a single one . Datasellers can choose to offer a redundancy bigger than the minimum redundancy level chosen by the app owner.

Segments are stored in distributed hashtables that are implemented in such a way that they can be queried by standard SQL queries on static and read-only data.

LSN data clients transform standard SQL schema and SQL datatables content into replicable distributed hashmap stored in datanodes. The conversion is done using the following equivalence:

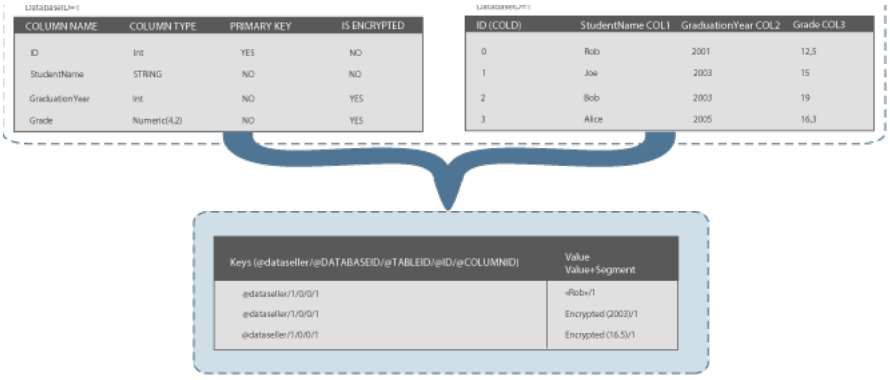

Figure 6: Datanodes storage structure

3.2 Storage contracts

Storage contracts are direct agreements between datanodes and data sellers. When the data seller initiates a storage contract, he locks the payment for the datanode. The data seller specifies the parameters of the storage contract:

- A valid application owner with its associated application contract
- The stored datatable merkle root
- The total size of the data table
- An XLC collateral to be paid by each datanode
- Proof of storage challenge frequency (such as in the SIA platform [4])
- Storage contract parameters: duration, due payment (such as in the SIA platform [4])

Beside the data "proof of integrity", the LSN infrastructure ensures a "Proof of storage" through a periodical challenge that all datanodes must pass in order to receive storage contract payments. The proof of storage algorithm is based on the retrieval of a random data chunk hash (a data chunk is a standard sized piece of the stored datatable, which is not to be confounded with datatable segments that are blocks of the minimum size in the datanode design. The data chunk is smaller that a data segment). The randomness is based on the block.prevhash information; each datanode has a specific time window to provide the required information.

For each successful challenge, the datanode is rewarded on its locked reward account with a specific "challenge reward". If the datanode is not successful in passing

a challenge, the corresponding reward is sent to the LNDO contract (in order to disincentivize DOS attacks). Each time a datanode fails a challenge, a transaction occurs from its storage collateral XLC amount to the LNDO contract. If the datanode fails to meet a specific number of challenges (to be specified by the dataseller), the corresponding contract terminates and the collateral is transferred to the LNDO. The datanode availability will then automatically be set to zero. If, on the contrary, the contract terminates successfully, the rewards are unlocked from the locked reward account and and sent to the datanode along with the initial collateral.

Both the application owner and the data seller define minimal requirements (redundancy and segment size) for data storage. Storage contracts store hashes of all datatables segments, making it costly to have too small segments (a segment size ranges from 1 to 128 megabytes). A balance has to be chosen between download time and data cost. For a standard 7-time redundancy ratio, and a 100 GB datatable, the estimated cost versus the segment size is detailed on the following graph:

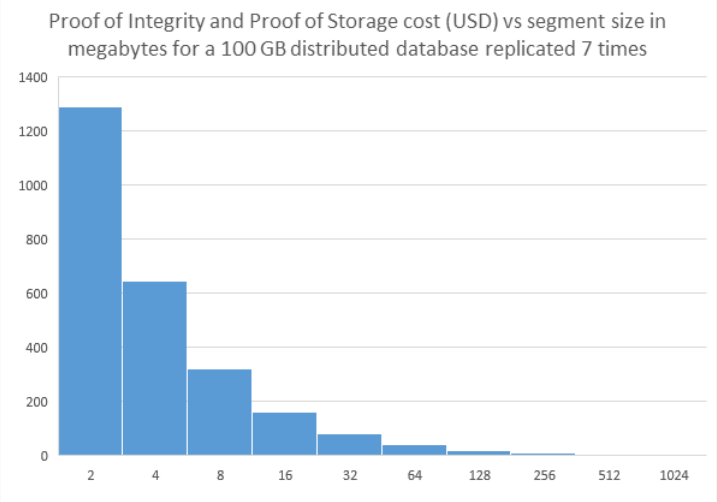

Figure 7: Proof of integrity and proof of storage cost vs segment size (in mb) for 100GB distributed database replicated 7 times

The segment size is parameterized for each datatable.

3.3 Datanodes management contract

Data sellers can deploy several storage contracts through a datanode management contract. The contract checks if the data seller is authorized by the application, select a default datanode with respect to its available storage, availability and amount of burn (which increases the chance of being selected to receive new data storing jobs, see section 7.3). The probability of a datanode to be selected by the algorithm is proportional to its amount of burn. Once a storage contract is deployed, it is added in a smart contract whitelist used by data buyers to verify the datanode honest behavior. Any update of the storage contract code can be done by the Ledgys team, with the approval of the LNDO.

3.3.1 Datanode registration

In order to be a candidate for storing Ledgys Distributed Network datatables, datanodes need to register in the Datanode Whitelist with:

- an IP address
- an available space for storage
- a maximum collateral availability
- a maximum contract duration accepted by the datanode
- an amount of XLC to "burn" (the amount will be added to the monthly reward).

For the registration to be valid, the burn must be superior to a minimal value minimalBurnValue, that will be modifiable by the LNDO.

If the parameters are all valid, the Datanode will be added to the whitelist.

3.3.2 Burning and datanode selection

At any time a registered datanode can call the burn function and send XLC to increase its burn value.

Datanodes are expected to burn part of their profit in order to maximise their probability of being selected. Additionally, a datanode can call the status function in order to set its availability to 0 or 1 (the function can't be called if it has been called recently).

3.3.3 Removing from whitelist

A datanode will be ejected from the whitelist if its availability is set at 0 for too long (according to a number of blocks set by the LNDO) or its burn value drops under minimalBurnValue.

If a datanode is removed from the list, it will lose all its associated burn value.

4 Application

4.1 Application data structure

The application owner can initialize the structure of all his affiliated data tables. Using a Ledgys schema specification language, the application owner will provide public description of the schemas through a Ledgys toolkit. The datatable schema specification hashes will be stored in the blockchain for databuyers to be able to check any received data according to the application owner schema constraints.

Those dataschema come with metadata information stored in clear text in the blockchain, that will help databuyers search for data providers through a standard keyword search.

4.2 Data storage parameters

The application owner defines technical parameters that must be agreed on by data sellers when publishing their data on LSN network.

- Challenge frequency : datanodes are frequently probed in order to assess their honest storage of immutable data. This parameter defines the frequency of the test.
- Minimum redundency: the data should be stored redundently in order to be available even in case of a datanode failure.
- XLC Collateral: Each datanode engage a collateral amount of XLC when entering in a storage contract, whose minimum amount is defined by the application owner.

4.3 Contracts

4.3.1 Applications Management Contract

An App Owner in the whitelist can call the createBuyContrat and/or createRestrictedBuyContract function to create its own standard buy contract. In the first one, anybody can buy the data whereas in the second one the appowner restricts the buy to a list of addresses. The parameters in these functions are appOwnerPercentage, Table format and description. appOwnerPercentage defines which percentage of each payment goes to the app owner, and which goes to the data seller (the remaining). Independently, a fixed percentage is granted to the LNDO.. The Table format parameter is a struct describing the column names and types and the description parameter is a string giving a hint on the identity of the appowner (to help the buyer identify him, see Buying Interface).

4.3.2 Individual application generated contract

Each individual application have their own custom contract generated by the Application Management Contract. In this generated contract, an App Owner can call the addSeller function to add the address of one of its data sellers. Datasellers will be stored as key value, where the key is the address of the data seller and the value a one string description (eg name) of the data seller. Dataseller can remove themselves from this list at any moment. They can also call a function to blacklist themselves from the generated contract (if they are currently registered as data seller). Dataseller can protect their data from chosen applications.

5 Data exchange infrastructure

5.1 The dataseller role

Data sellers define payment schemes through a local client based on the common datatable scheme defined by the application owner. They have the following options:

- Define a data price cell by cell
- Define a unique price by column
- Define a unique price for every encrypted data in the table
- Define a unique price for a specific data computation with specific search criteria

In order to simplify the definition of data prices, data seller clients use a custom scripting language that understands SQL queries on the data seller datatable.

Examples:

```
SET PRICE = 1 XLC IN (SELECT FruitPrice FROM Shop)
```

This query will set a price of 1 XLC for any fruit price in the shop.

```
SET PRICE = 10 XLC IN (SELECT * FROM AccountingTable
WHERE CustomerId IS_UNIQUE).   The IS_UNIQUE operator
is not a standard SQL operator, it has been added to
the LDN query language to define an homogeneous data
group in which each data item (with a unique column
and a unique row) will have the same price, 10 XLC.
```

This query will set a price of 10 XLC for any accounting data of a unique customer.

Custom algebraic operations can also be priced through the scripting language. The dataseller, can set a price to custom operations on the data, but to prevent any data items being used in the computation to be deciphered. The databuyer will be able to compute the data himself and request the correct key for his computation, thus being certified that the deciphered data is the result of a transparent computation.

5.2 The databuyer role

5.2.1 Querying data

The databuyer uses a SQL-compliant data client to query data on the LDN datanodes. Through this interface, the databuyer can select the relevant application and the list of affiliated datasellers, or could also perform a data search on all applications and all datasellers at the same time.

The client enables SQL search on chosen datatables schemas on the distributed network of datanodes, through the following process:

1. The data buyer broadcasts an SQL query to all datanodes. With his query is attached the hash of the datatable, as well as the application owner blockchain address.
2. All datanodes are looking up wether they store the queried table, perform the SQL query on the table and inform the data buyer of the search result - even if there were no data at all.
3. The data buyer randomly selects a datanode with positive results on the queried data. On the selected datanode, the databuyer performs a 4-step sanity control that if failing will lead to another datanode selection.
4. The databuyer 4-steps sanity check control on the selected datanode:
    - The selected datanode storage contract is valid (it is in the validStorageContract).
    - The dataseller address must be in the buycontract of the requested application owner. This ensures the selected datanode is officially working with the chosen application.
    - The datanode sends a signed message, including the random seed in its body, proving that they own the ethereum address in the previous application owner smart contract. This is a strong identity proof.
    - The number of successful challenges is strictly above 1. This control that the datanode is not a newcomer and has already proven to store the queried datatable.

The databuyer client is directly connected to the blockchain to ensure that no trust is needed. While querying data on the LDN, the databuyer verify all applications and datasellers ID by comparing their IDs with related blockchain downloaded information.

5.2.2 Paying for upload and data keys

The databuyer downloads the data from datanodes and pays each relevant datanode for their uploads; this payment disincentive segment withholding attack (see section 9.2). A micropayment scheme is set up to pay data segment by segment, after checking the integrity of each received segment. To do so, the data buyer compares a hash computed locally with the hash stored in the blockchain, in the storage contract associated to the datanode. Once all the relevant data segments have been downloaded, the data buyer pays for the data decryption keys. The payment transaction has 3 recipients : the data seller, the application owner and the LNDO (the LSN central authority).

Depending on his SQL query, the databuyer will either buy decryption piece to decrypt data cell by cell using the buyCells function, or will buy the result of a specific computation he performed on the encrypted data using the buyComputationResult function. (In either case, all key holders (Ledgys, the application owner or the

dataseller) will send a key to decrypt the requested value only. In the case of the buyComputationResult function, the requested key will be computed by the parties.

6 Cryptographic keys sharing and exchanges

We implement a datatable driven encryption scheme where all data are organized in datatables, such as in relation databases. In those datatables, some of the columns are not encrypted, and often represent searchable items or business data keys: unique identifiers, public names or places, dates, public business citerions.

6.1 Encryption framework

Our chosen algorithm is a public-key encryption scheme which relies on the Diffie–Hellman key exchange. We use a standard elliptic curve $\mathscr{C}$ on a finite field, with a group $\langle g \rangle$ of prime order on $\mathscr{C}$.

The data seller chooses a private key $x_j \in Z_q$ for each encrypted column of a table, as well as a secondary key $r_i \in Z_q$ for each row.

Then there is two possibilities depending on what the column contains :

- If a value $m_{i,j}$ is not subject to calculations, we encrypt it as

$$d_{i,j} = m_{i,j} \oplus \mathscr{H}(x_2 r_i \cdot g)$$

  with $\oplus$ the bit-to-bit XOR function and $\mathscr{H} : \mathscr{C} \to \{0;1\}^{512}$ a hashing function.

- If we want to perform calculations on it, we consider $m_{i,j}$ as the representation of a positive integer and encrypt it as

$$d_{i,j} = m_{i,j} \cdot g + x_3 r_i \cdot g \in \langle g \rangle \ .$$

The data part d_{ij} of the cypher is kept in the datanodes.

The keys r_i are only known by the dataseller, the application owner and Ledgys Distributed Network servers.

In case of a data query on encrypted data, we have several options:

- If the data is encrypted with a checksum from $\mathscr{H}$, we get back the original message with

$$d_{i,j} + \mathscr{H}(x_j r_i \cdot g) = m_{i,j} \oplus \mathscr{H}(x_j r_i \cdot g) \oplus \mathscr{H}(x_j r_i \cdot g) = m_{i,j}$$

- If the data is encrypted as a group element we can get

$$m_{ij} \cdot g = d_{i,j} - x_j r_i \cdot g$$

Provided that $m_{i,j}$ is not too big, it is possible to retrieve $m_{i,j}$ from $m_{ij} \cdot g$ thanks to the λ-Pollard algorithm also known as kangaroo algorithm.

This is a probabilistic algorithm based on the birthday paradox with an average complexity of $O(\sqrt{b})$ where b is the biggest value that $m_{i,j}$ could take [3].

The security of the algorithm comes from the fact that it is computationally difficult to find k from the values g and $k \cdot g$ on an appropriate elliptic curve when k is big enough.

6.2 The 3-party key sharing scheme

Initially, datas are encrypted by data sellers, who then split their private keys x_j into several tokens using Shamir's Secret Sharing (SSS).

SSS is basically using polynomial interpolation with N points on a M-degree polynomial ($M < N$) to solve a secret, so that we need $M + 1$ of the N points.

At the launch of the platform, this parameters will be set at $M = 1$ and $N = 3$.

Each single column of a database will be encoded using a specific key, and each of its key will be split in 3 components.

Each key holder (the dataseller, the application owner and Ledgys) will keep one of those components.

For the key distribution, we are using a custom Library using IPFS[6] and Elliptic Curve encryption to define a shared secret.

This approach provides mutual authentication, data integrity control, as well as data secrecy.

Using our chosen El Gamal encryption scheme, our encryption key secret sharing, our ability to reconstruct partial keys on demand in order to serve only the decryption of a single value or a single computation, the dataseller ensures himself that the databuyer won't be able to decrypt more than what he's been allowed to, even if he already downloaded the whole encrypted database from the datanodes.

On his side, the databuyer only need to trust the three encryption key holders for correctly storing and sending their partial decryption keys: the databuyer database can be local, the computation can also be done locally.

Besides, each partial key received from a key holder is also packaged with a cryptographic proof that the sent decryption key is correct (whether it is for a single data value decryption or for a computation decryption).

Two of the values $s_{j,1}, s_{j,2}, s_{j,3}$ are enough to calculate x, but we don't want the buyer to find this value. Thus, for every message $m_{i,j}$, the values $s_{j,1}r_i \cdot g$, $s_{j,2}r_i \cdot g$ and $s_{j,3}r_i \cdot g$ are sent to the buyer who can computes $xr_i \cdot g$ with two of those values.

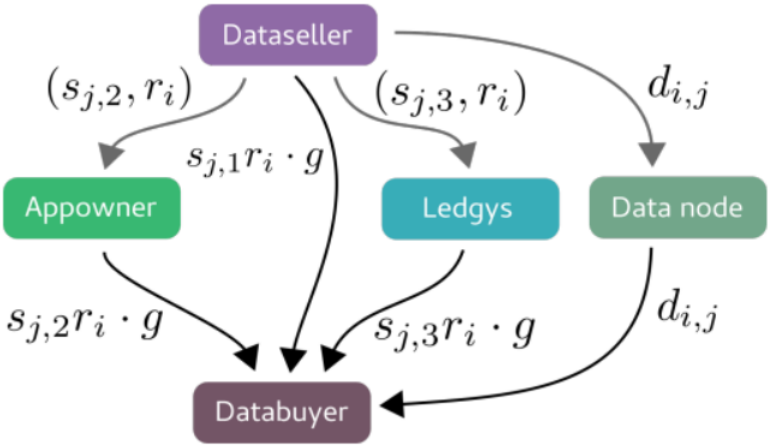

Figure 1 – Keys Distribution

Two of the values $s_{j,1}, s_{j,2}, s_{j,3}$ are enough to calculate x, but we don't want the buyer to find this value. Thus, for every message $m_{i,j}$, the values $s_{j,1}r_i \cdot g$, $s_{j,2}r_i \cdot g$ and $s_{j,3}r_i \cdot g$ are sent to the buyer who can computes $xr_i \cdot g$ with two of those values.

Figure 1: Keys Distribution

6.3 Calculations

Data buyers are not just interested in the raw data that are on the datanodes but may also want to perform calculations on it. Let us say we want to know $a = \sum_{(i,j)\in E} \alpha_{i,j} m_{i,j}$ with E a set of coordinates, $n = Card(E)$ and $(\alpha_{i,j})_{(i,j)} \in Q^n$ the set of coefficients.

Let w be the an integer such as $\forall (i,j) \in E, w\alpha_{i,j} \in Z$, and $\left(\beta_{i,j}\right)_{(i,j)\in E} = w \times \left(\alpha_{i,j}\right)_{(i,j)\in E}$.

The data buyer calculates

$$d = \left(\sum_{(i,j)\in E} \beta i, j m_{i,j}\right) \cdot g + \left(\sum_{(i,j)\in E} \beta i, j x_j r_i\right) \cdot g \ .$$

Each key holder, with number σ sends the key $K_\sigma = \left(\sum_{(i,j)\in E} \beta_{i,j} s_{j,\sigma} r_i\right) \cdot g$.

Let σ and τ be two key-holders number. Then the data buyer computes the following :

$$\lambda_\sigma K_\sigma + \lambda_\tau K_\tau = \left(\sum_{(i,j)\in E} \beta_{i,j} x_j r_i\right) \cdot g$$

We have $d - (\lambda_\sigma K_\sigma + \lambda_\tau K_\tau) = wa \cdot g$ and are able to retrieve a thanks to the λ-Pollard algorithm.

6.4 Key transfers and key holders' responsibilities

Any payment done by data buyers is notified to the encryption key holders, along with either the data cell coordinate to be deciphered, or a computation formula. Through public Ethereum smart contract logic, the key holders decide to distribute the decryption key for the requested data cell or computation formula, allowing a full control on which database facets should be revealed. The key distribution purchase act is decided through a public smart contract and thus can easily be audited; physical partial decryption pieces distribution occur on the peer to peer LDN in an automatic server to server communication.

If some computation on encrypted data is finally authorized, the 3 key holders are responsible for computing their part of the data buyer computation's decryption piece. This computed decryption piece is itself a 3-party secret sharing key, that can only be reconstructed if two of the key holders have replied to the data buyer's request.

Our protocol does not require any key holder (the dataseller, the application owner and Ledgys) to store any of the business data. It only needs to keep a pricing scheme associated to each data cell value and openly published by the dataseller, with the golden rule that the most expensive datacell price will always prevail when computing a formula on datacells with distinct price.

Thus, Ledgys is reponsible for maintaining a global repository of public information about all the pricing scheme of all dataseller databases from all Ledgys' applications, whereas each application owner will maintain a local repository of only its own dataseller pricing scheme. The data seller has the responsability to keep its pricing scheme public and available.

6.4.1 Calling for datacells values

The databuyer will call the buyCell (or buyCells if there are several datacells) smart contract with the following parameters:

- His IP: This is required to receive the decryption keys once the payment will be proved successful
- The hash of the requested datatable
- The dataseller blockchain address
- The indices of datacells to be decrypted
- The estimated price of the datarequest. The databuyer client can connect to pricing schèmes off blockchain and compute the estimated query price locally.

Before returning any key to the dataseller, each key holder will perform the following actions:

- Check if the dataseller is registered for the current application
- Check if the requested datatable is a valid one for the current application
- Check if the request price is correctly computed regarding the dataseller pricing scheme

6.4.2 Calling for a computation decryption

The parameters are the same, except for an extra parameter describing the computation formula. The key holders compute the corresponding decryption keys, as well as a cryptographic proof that the reconstruction pieces are correctly produced.

7 Ledgys Network Distributed Organisation

Datanode are administered through a decentralized governance thanks to public smart contract rules. Applications code cannot follow the same goverannce rules; Ledgys Distributed Network introduce moderation processes to track applications content evolution, through a distributed monitoring organization, the LNDO (Ledgys Network Distributed Organization).

The LNDO not only moderate applications content, but also the whole Ledgys network smart contract code. The origination of the LDNO organization will occur though the XLS token crowdsale, whose usage will be to give voting rights and rewarding benefits.

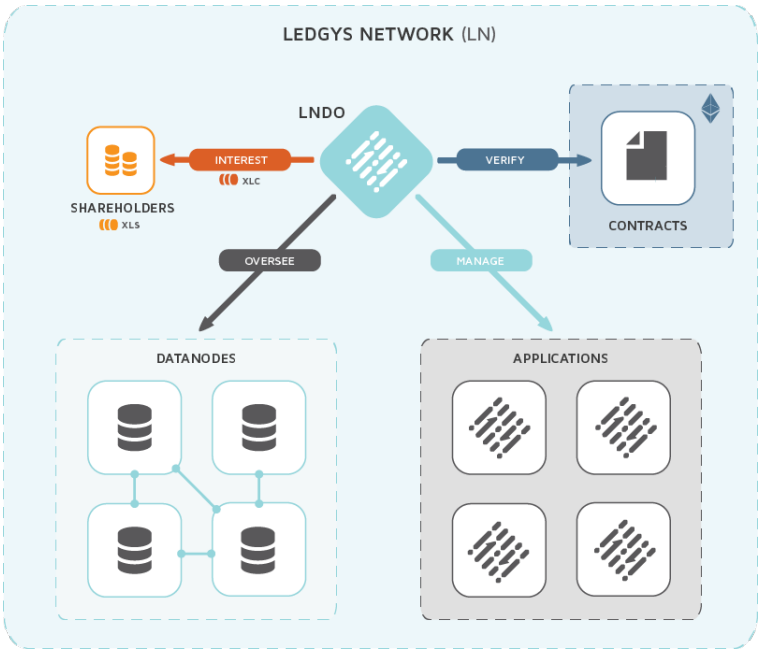

Figure 9: The Ledgys Network Distributed Organization roles

The LNDO will oversee the network by accepting, warning and removing applications from a whitelist defined in the LNDO contract.

The organization will also have the task of monitoring Ledgys, by approving the code and parameters changes of any smart contracts administered by Ledgys.

7.1 LNDO Shares

The XLS token will be the cornerstone of the Ledgys Network Decentralized Organization, the holders of the XLS shares will be able to code changes, to monitor applications content and behavior. In exchange of their monitoring effort, shareholders will receive a fee collected by the LNDO.

To be able to benefit from the LNDO rewards and to participate in voting, XLS shareholders will need to engage their XLS for a certain amount of time. In Ledgys Distributed Network, the time measurement is done in Ethereum blocks:

- a vote needs a minimum of 7 days commitment
- rewards will be given each month

Locked XLS shared will be monitored in a Shares management contract, that will maintain an XLS balance list, one balance per shareholder. For each balance, a reserve amount defines the amount of XLS shares locked, and the reserved block is a measure of time for future rewards and votes. To commit XLS to the reserve list, shareholders will need to use the "engage(amount)" function of the LNDO contract.

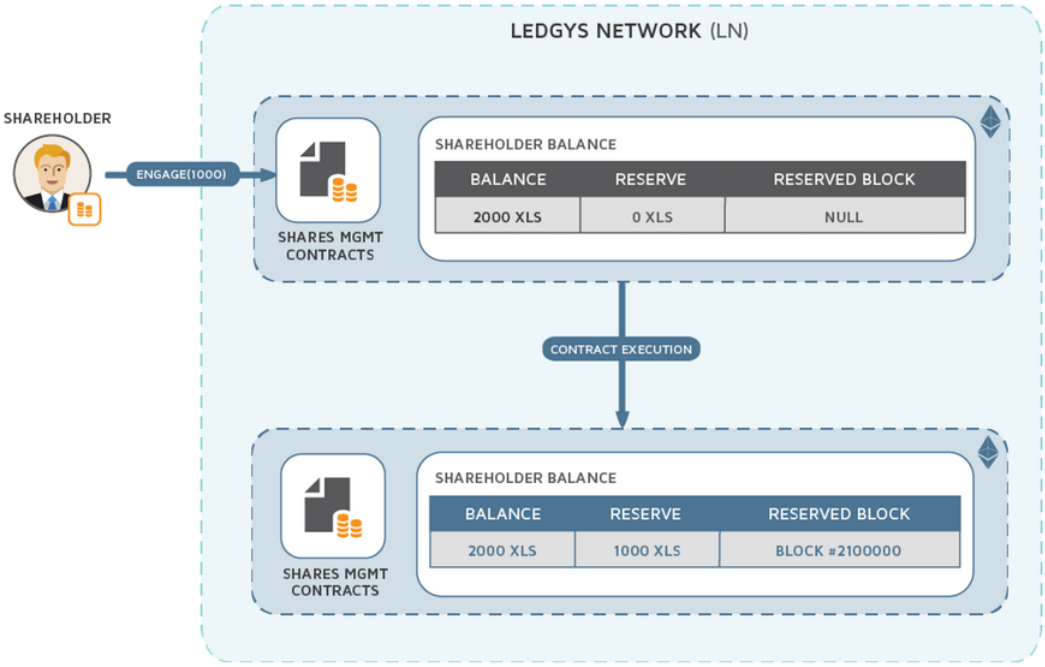

Figure 10: Engaging Ledgys Distributed Network Shares funds

Monthly reward

The reward will be given each 30.436875 days (average for a month), i.e. each 187800 blocks for Ethereum time, computed using an average of 14 seconds / blocks (in the homestead release). Practically, the smart contract will reuse the Solidity day Time Unit, and the block references are only here for the algorithm understanding.

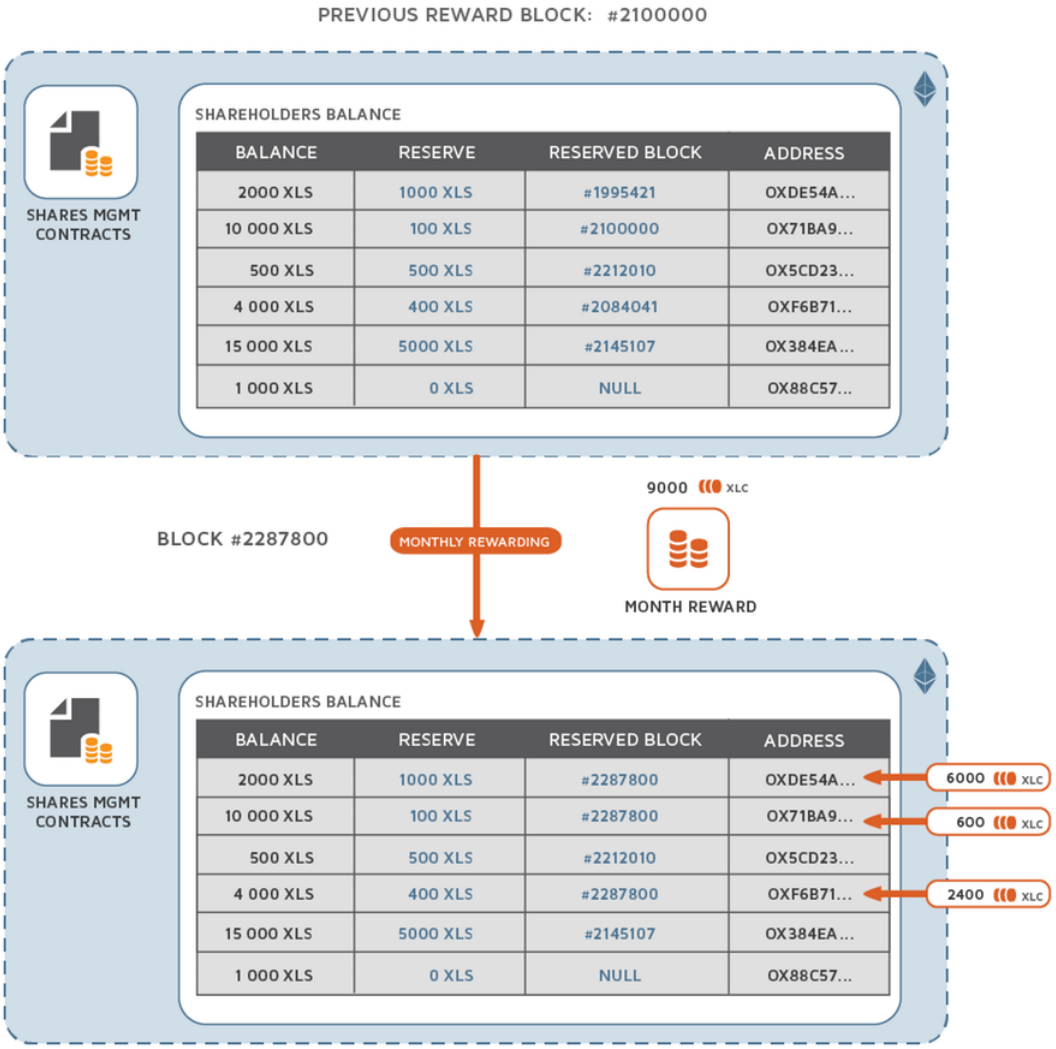

Figure 11: Monthly rewardings for engaged XLS

The voting process

To participate in a vote, a shareholder will need to lock his XLS for a minimum of 1 week, i.e. 43200 blocks before the current block (the current block where the vote function call is occuring). To engage part of his shares, the XLS holder can call the engage function; however the XLS holder must disengage all his locked shares and cannot disengage them partially.

When added new shares in his reserve, the shareholder reserved block is modified using the following formula:

$$RBlock + \frac{Reserve_{New}}{Reserve_{New} + Reserve_{Old}} x(LastBlock - RBlock)$$

Example:

- Total Amount : 10 000 XLS, Reserve : 1 000 XLS, RBlock : 2'100'000
- The shareholder wants to add 1000 XLS to the reserve, at block 2'150'000
- The new Reserve amount will be 2000 XLS
- The New RBlock becomes $50000/2 = 25000$ -> 2'125'000

7.2 Monitoring the applications through the LNDO

The LNDO will manage application through a whitelist, implemented in the LNDO contract. To be accepted in the whitelist, the application will need to receive an approval vote. On the contrary, applications can also be removed from the list if shareholders believe the application does not conform to ledgys network.

7.2.1 The application registration process

To register a new application to the Ledgys Distributed Network, the application owner must send an application proposal to the LNDO, through the apply function of the LNDO contract. The parameters to register an app are:

- The application name
- The application domain name or link
- The required application fee on all transactions occuring in the application scope
- The Ethereum address of the owner

An application registration fee is required to avoid a flooding of empty registrations, this fee is added to the LNDO fees account. The application registration is done in a two weeks period, and the application is accepted once a 20 percent quorum is reached; at that stage only the application owner will be authorized to generate its unique application smart contract through the application management smart contract. A pending list is created to keep track of applications not yet generated. The default application parameters are set periodically by the LNDO; the generated

contract owner will be the application owner. By default, the new contract will be added in the LNDO applications white list; the application owner will then be able to launch its own data marketplace.

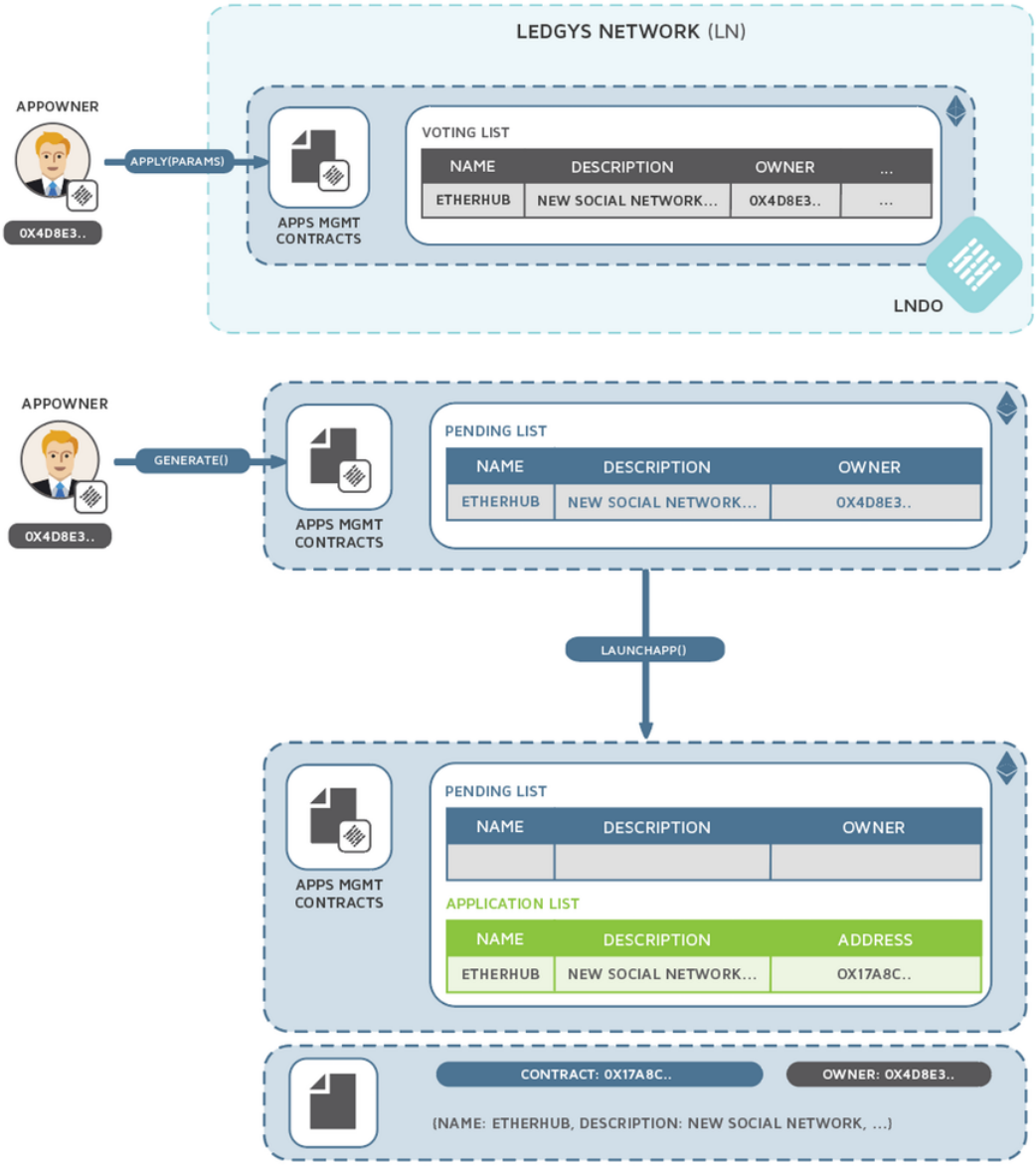

Figure 12: The application approval process

7.2.2 Warning and removing applications

Wrong doers can be warned through auditable and public messages explaining the warning reason, at a maximum frequency of one warning per week. A warning requires a quorum of 10 percent. After 3 warning, the application is removed from

the application whitelist and won't be able to operate on the Ledgys Distributed Network. With a 20 percent quorum, a warning can also be removed from an application. Those quorum levels ensure that removing a wrong doing application can be fast, but an unfair exclusion by a minority group can be removed through a broader community action.

If a warning reaches a 40 percent quorum, the application will automatically be excluded from the whitelist.

7.3 Datanodes supervision

Datanodes are directly managed by a whitelist managed by the datanode management contract; however the LNDO cannot micromanage the datanodes, this is an automatic process performed by a smart contract according to datanode regularity, performance and honesty.

8 Global Ledgys Distributed Network architecture

The whole LSN network mechanisms described previously can be synthesized in the following diagram:

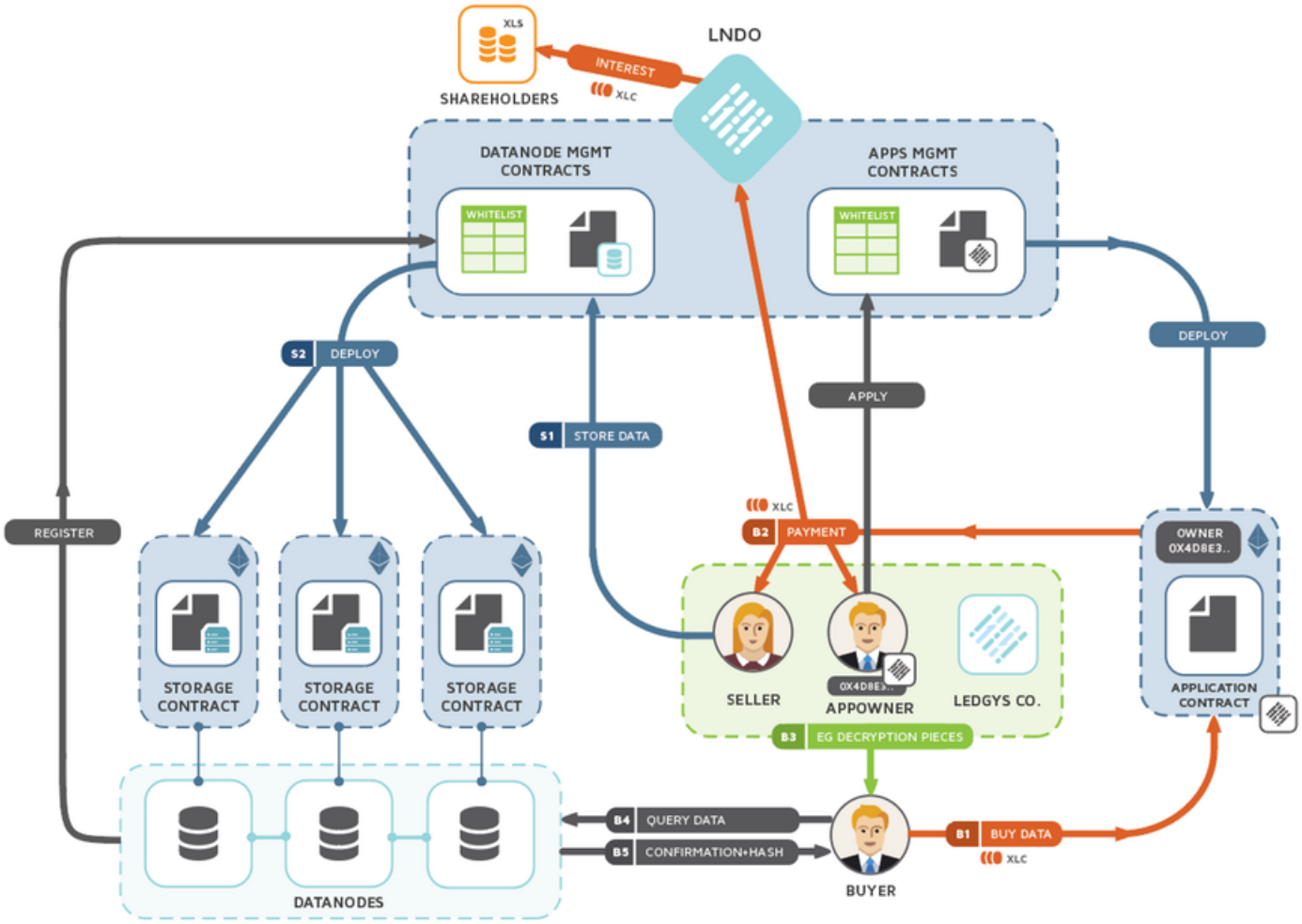

Figure 12: Detailed overview of Ledgys Distributed Network

9 Sybil attacks and bad actors attacks

The LSN networks have been designed to prevent common attacks on distributed systems.

9.1 The Sybil attack

In a Sybil Attack, the attacker forges a false identity to misbehave in a distributed system. In the LSN case, a malicious datanode could pretend to be many in order to be frequently selected by the LSN selection algorithm, while declining at the same time every storage contract proposal in order not to contribute to any collective storage effort. For example, a malicious datanode could pretend to be many in order to be frequently selected by the selection algorithm, and decline every storage contract, considerably lowering the reliability of the network.

The burn mechanism makes it very expensive to conduct such attacks, because the selection algorithm does not care about the number of addresses, only about the relative burn proportion of each address. An attacker trying to be selected as often as the rest of the network will thus need to burn as many coins as the rest of the network combined. The attacker could be inclined to do it in order to be selected to store the whole redundancy of one database, basically taking hostage the corresponding data seller. Data sellers can protect themselves by increasing the redundancy of their database. A 10x redundancy is safe from this kind of attack as long as one datanode hasn't burn 9x as many coins as the rest of the network combined.

Renters storing at 10x redundancy are safe unless the attacker looks like 91 percent of the network, requiring the attacker to burn 9x as many coins as the rest of the network combined to be successful. It is up to data sellers to find the adequate balance between cost and redundancy, with respect to their security requirements.

9.2 Segment witholding attack

While storage contracts ensure that the data is stored and not altered, they don't provide any incentive for data upload. It means that with storage contracts only, nothing compels the datanode to upload data upon request. To solve this issue, we use a micropayment scheme as incentive for segment upload.

Let us imagine that a data buyer wishes to download 10 segments from a datanode. To do so, he opens a micropayment channel and pays 1/10th of the total each time he receives a valid segment. Each micropayment is made in advance of the upload, and the datanode is incentivized to upload a valid segment not to miss the next payment.

We still have two problems. First, there is apparently no incentive for the datanode to provide the last micropayment, since the payment is made prior to each segment upload. Second, for 1-segment request, no micropayment channel is opened and

we only have one direct transfer prior to segment upload, thus loosing the future payment incentive.

We address these problems using the combination of redundancy and local blacklisting. In both situations, if the datanode fails to send a segment after a payment as been made, the data buyer can add it to his local blacklist and retrieve a redundant copy of the segment from an other datanode. The faulty datanode will thus not receive any payment from this buyer in the future. In the Ledgys network, buyers will probably be regular buyers of a specific type of data. It means that datanodes will likely be frequently requested from the same buyers. Being blacklisted by this buyers is thus substantially economically penalizing.

9.3 Multicast autoconfirmation

A query is multicasted to every datanode in the validStorageContract whitelist. It means that a datanode who managed to be selected for a storage contract could potentially answer positively every query he receives. He wouldn't care about local blacklisting and just look for short term profit. This is what we call a multicast autoconfirmation attack (terme de Gautier). First, this attack is expensive to repeat. Because of the burn mechanism, a datanode whishing to repeat this attack by re-entering the whitelist with a different address to 'reset' its blacklisting would have to burn each time a fair amount of coin to have a good chance of being selected.

Then, this attack can be profitable only if the number of query is very high and come from a great diversity of data buyers. We will thus consider a later stage of the development of the Ledgys platform, where we would be in such situation. At this point, the last layer of protection is the 4 steps verification described in Query. Basically, we make sure that the datanode is currently storing a table in the ecosystem of a specific app owner (ie belonging to one of his affiliated data seller).

If the buyer knows the data seller it intends to buy from, it is even possible to make sure that the datanode stores a datable belonging to a specific data seller. We thus largely reduce the scale of the multicast autoconfirmation attack. Datanodes can only capture the micropayment of datanodes storing table of the same app/-dataseller. In an ecosystem where there are lots of apps (our predicate), it is a lot less rewarding to conduct such attacks. Furthermore, in this case, it becomes way more penalizing to be blacklisted, as buyers often interact in bounded ecosystem, meaning that they often buy from the same app/ data seller. Thus, we can predict that it will not be overall profitable for datanode to perfom multicast autoconfirmation attacks.

9.4 DOS attack

The DN Hosts could be subjet to DOS attacks. It is their responsibility to protect them from such attacks. The reward mechanism of storage contract disincentivize a DOS attack to prevent datanode from passing challenge. The fact that the reward for an unsuccessful challenge is burnt instead of refunded suppresses any economic

incentive for data seller to DOS attack datanodes in order to benefit for cheaper storage.

9.5 Trust (Datanode integrity, Keys, Dataseller attack other data seller)

While trust is not an attack per se, it is a major concern of many parties that engage into an ecosystem. In the Ledgys network, different levels of trust are required for each parties. Datanodes evolve in a complete trustless environment, which means that they need not trust any other parties. Data buyers need only trust the key holders (the Ledgys Company, the application owners and the data sellers) for sending the decryption keys.

They need not trust them for the validity of the pieces they send, or trust any other party in the ecosystem. The risk of pieces withholding from any of the key holders is very thin, since they are publicly known and economically vested in the ecosystem. Conducting withholding attacks would deeply hurt their reputation and their economic profitability. The risk is also mitigated by the fact that every payment is done data cell by datacell, and represent inherently small profits compared to upcoming future revenues.

Additionally, both the Ledgys company and all application owners are public actors with vested interests in the data marketplace ecosystem. The risk of an actor collusion to resell the database on other markets than the Ledgys Distributed Network is limited by the reputation risk would the exploit be revealed.

References

[1] Eric S. Raymond “The Cathedral and the Bazaar” (1999).

[2] Gavin Wood, “Ethereum: a secure decentralised generalised transaction ledger” (2015). `http://gavwood.com/paper.pdf`

[3] Teske, Edlyn, “Computing discrete logarithms with the parallelized kangaroo method”, Discrete Applied Mathematics (2003).

[4] Vorick, Champine, “Sia: Simple Decentralized Storage” (2014). `https://sia.tech/assets/globals/sia.pdf`

[5] Mao, Wenbo“Modern Cryptography”, Modern Cryptography (2004).

[6] Juan Benet, IPFS - Content Addressed, Versioned, P2P File System `https://ipfs.io/`