# BROKERAGE AND MARKET PLATFORM FOR PERSONAL DATA

## *D5.3 Initial KRAKEN marketplace integrated architecture document*

www.krakenh2020.eu

This project has received funding from the European Union's Horizon 2020 (H2020) research and innovation programme under the Grant Agreement no 871473

# D5.3 Initial KRAKEN marketplace integrated architecture document

| | |
|---|---|
| **Grant agreement** | 871473 |
| **Work Package Leader** | LYNKEUS |
| **Author(s)** | Davide Zaccagnini, Minos Garofalakis, Alexandros Tragkas (LYNKEUS) |
| **Contributors** | Rob Holmes (TEX), Donato Pellegrino (TEX), Davide Porro (INFOCERT), Angel Palomares (ATOS), Tilen Marc (XLAB), George Pikramenos, Konstantinos Kechagias, Stefan More (TUG), Sebastian Ramacher (AIT), Karl Koch (TUG) |
| **Reviewer(s)** | Stephan Krenn (AIT), Giancarlo Degani (INFOCERT) |
| **Version** | Final |
| **Due Date** | 30/06/2021 |
| **Submission Date** | 30/06/2021 |
| **Dissemination Level** | Public |

**Release History**

| Version | Date | Description | Released by |
|---|---|---|---|
| V0.1 | 04/07/2021 | Draft ToC | Davide Zaccagnini |
| V0.1 | 06/03/2021 | Complete Draft including contributions from TEX, AIT, TUG, ATOS, ICERT | Davide Zaccagnini |
| V0.2 | 06/22/2021 | Reviewed by AIT and ICERT, changes incorporated. Sent to the Coordinator for submission | Davide Zaccagnini |
| V1.0 | 30/06/2021 | Submitted version | Atos |

# Table of Contents

# List of Tables

# List of Figures

# List of Acronyms

| Acronym | Description |
|---|---|
| CA | Consortium Agreement |
| WP | Work Package |
| BD | Batch data |
| CSR | Certificate Signing Request |
| DAC | Data Access Control |
| DID | Distributed Identifier |
| DLT | Distributed Ledger Technology |
| GDPR | General Data Protection Regulation |
| HLF | Hyperledger Fabric |
| KRAKEN | HORIZON 2020 project KRAKEN (Brokerage and market platform for personal data) |
| MPC | Multiparty Computation |
| MSP | Membership Service Provider |
| SC | Service Certificate |
| SD | Streaming Data |
| SIS | Student Information System |
| SM | Smart Contract |
| SMPC | Secure Multiparty Computation |
| SSI | Self-Sovereign Identity |
| SSI SDK | SSI Software Development Kit |
| TPS | Transactions per Second |
| VC | Verified Credential |
| UI | User Interface |

# Executive Summary

The KRAKEN marketplace is designed to allow the highest level of control over personal and institutional data assets by their legitimate subjects and/or controllers, i.e. data sellers, and the most efficient and costs-effective way to access such resources by buyers, i.e. data users. The design is strongly focused on decentralized, integrated architectures in the areas of identity management, data access permissioning and transaction management, rather than centralized control by third parties. The architecture leverages state of the art trends in the area of distributed, peer to peer networks and new security paradigms. Multiple permissioned blockchain's are interconnected and integrated with both application layers and data protection infrastructures. The current scope defines all key aspect of the marketplace architecture. Areas currently being designed include the Kraken mobile app and detailed integration of the Secure Multiparty Computation system for distributed analytics.

# 1 Introduction

## 1.1 Purpose of the document

This deliverable 5.3 describes the architecture of the KRAKEN marketplace an integrated set of infrastructures implementing two pilot use cases for the exchange of biomedical and educational data.

This report serves to inform internal and external stakeholders on the inner workings of the platform, also for communication purposes, and to base further designs of components and functionalities for the remainder of the project, including the definition of interfaces with external systems, the KRAKEN mobile app, of the verified credential systems and the SMPC infrastructure. The integration of all these additional components has been broadly scoped and it's being detailed during the current period.

This document draws from multiple other sources including D2.2 Intermediate Kraken architecture, D2.6 Marketplace Technical Specification and D7.2 Ethical and legal requirement specification

## 1.2 Structure of the document

The document is structured in eight (8) main sections dealing with respective components on the KRAKEN marketplace architecture. Outstanding issues and strategic directions for the following period of the project are discussed at the end.

# 2 The KRAKEN Data marketplace

The KRAKEN marketplace architecture consists of three main functional areas, as indicated in the diagram below. These are:

1. The permissioning layer where data access is controlled leveraging the Lynkeus Hyperledger Fabric Blockchain
2. The data access layer providing multiple infrastructures and methods allowing secure and private access to data products including the SMPC system, the TEX streaming data infrastructure and the batch data exchange system developed in the first period of the project.
3. The transaction management layer featuring technologies supporting user workflows, payments and fulfillment, mostly leveraging the TEX, Streamr marketplace.

These three layers are functionally integrated to first grant or deny data access based on the legally binding rights, then provide such access on three different modalities (SMPC, batch or streaming), and then monitoring the fulfillment of all key transaction steps.

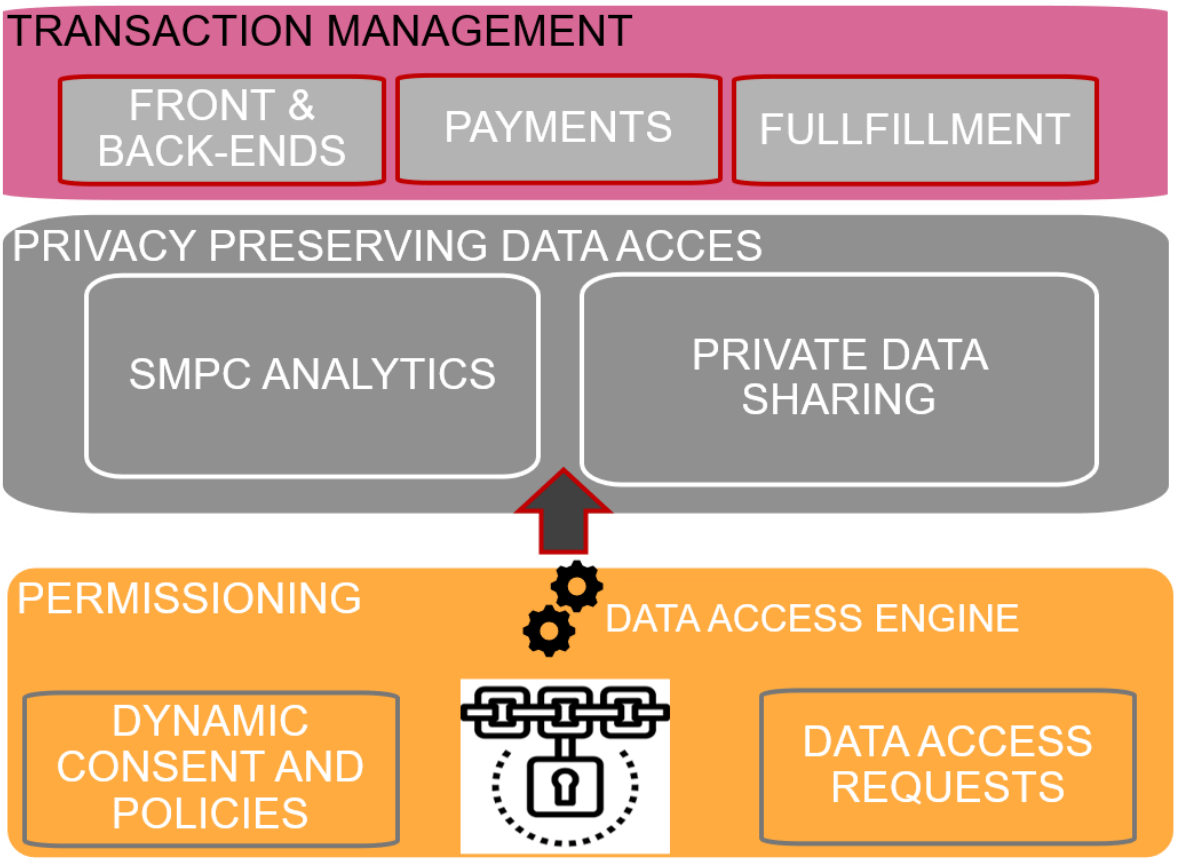


**Figure 1 The KRAKEN Technology Stack**

From a functional perspective (see below) the Marketplace API, i.e. the back-end, connects both the desktop and the mobile apps to all other components. In particular, SSI identities are passed to the data access layer on which permissions are computed. Positive access decisions are passed through the API module to the data access layer and then to the xDai payment and fulfilment system.

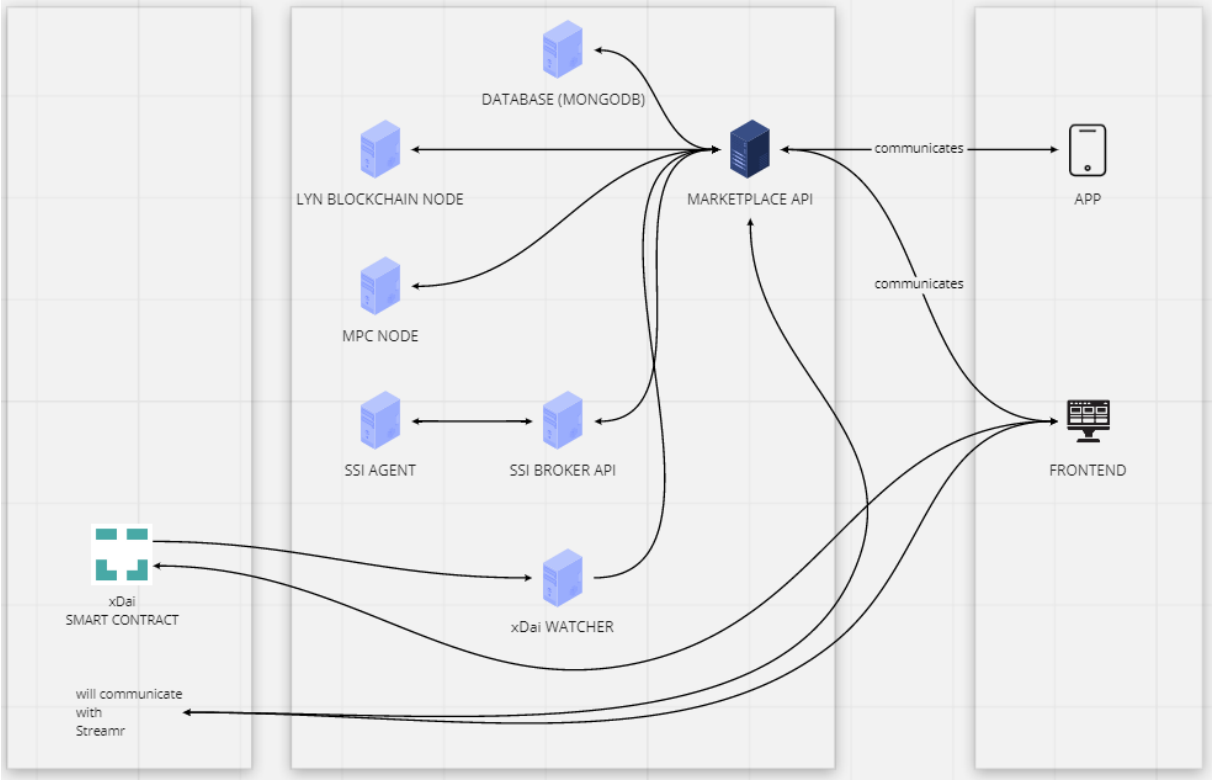


**Figure 2 Architectural diagram**

This architecture is a result of parallel and iterative design efforts to link multiple modules and infrastructures, each at different stages of technological maturity. These include the Streamr marketplace, the Lynkeus data access layer, the Self Sovereign Identity system, the Verified Credentials infrastructure and the data protection layer, which is itself composed of the SMPC system and ad-hoc data protection modules (ex. Batch data encryption). The guiding principle of such design, and indeed of the project itself, is to implement true decentralisation throughout the marketplace and the overall platform itself while at the same time providing the highest level of privacy protection for its users and the data they will exchange. Compliance with national and European privacy laws has been, in this view, a key concern in the development of this integrated system, In strict conjunction with WP7, intermediate designs, and implemented components with their integrations, were systematically reviewed from a legal and ethical stand point following a privacy by design approach. This work is still ongoing as new modules and UI extensions are added with the final aim at automating the enforcement, by the platform itself, of legally and ethically binding terms users can enforce for the temporarily access and process of personal data for predefined purposes.

From a functional standpoint the architecture is divided in three areas:

1. **Permission management**, mostly implemented by the Lynkeus Hyper Ledger Fabric blockchain in conjunction with the SSI system for the identification, authentication and credentialing of both individual and organisational users.

2. **Data protection layer**, which implements a variety of data security and privacy preserving modules and includes the Secure Multi Party Computation system employed for both distributed data analytics and encryption keys sharing mechanism, in addition to the standard data protection functionalities such as encryption at rest and transaction of batch and streaming data assets.

3. **Data transaction management**, mostly implemented through the Streamr marketplace technology which provides both user-facing and back-end functionalities, such as UIs,

payments execution and control, secure transfers of streaming data, data product visualization and more.

# 3 Permission Management

## 3.1 Distributed, peer-to-peer data access control

The permission management layer is tasked with controlling what users have legitimate, compliant access to what data products, based on the policies set by data subjects or controllers to share their data and it is based on the Hyperledger Fabric blockchain developed by Lynkeus over the course of multiple EU-funded research projects, The system implements a set of healthcare specific data access permission criteria which are set by both individuals and organisations interested in sharing their data in the network and has been extended to cover educational data transactions. A filtering algorithm running on the blockchain checks data access requests against such criteria so to allow a specific data access request to be fulfilled, or not, based on legal parameters (GDPR and national regulations), local and individual policies specified by the data controller or subject, such as the intended uses of the data specified in informed consent issued by the data subject(s).

## 3.2 Blockchain technologies and new data ecosystems

Blockchain, a type of Distributed Ledger Technology (DLT), is a sequence of blocks of data records stored in a distributed, transparent, and immutable infrastructure. Blockchain utilizes practically unbreakable cryptographic schemes, which enforce privacy while decentralizing the decision-making process through a set of agreed upon network policies (see below), not to be confused with data access policies set by data subjects and controllers. Applications built on top of blockchain can therefore access a trust-less transaction management system that enforces the compliant execution of such transactions, disincentivizing malicious actors.

## 3.3 Hyperledger Fabric

Hyperledger is a collaboration between the Linux Foundation and big tech organizations including IBM, Intel, Consensys, and others. Hyperledger Fabric (HLF), in particular, provides a highly versatile modular infrastructure for enterprise use cases. Providing tools to implement a permissioned network, Fabric allows for pluggable consensus mechanisms such as those KRAKEN implements for gathering informed consent from user regarding the use of their data. Its underlying architecture gives organizations the ability to customize multiple aspects of the network in terms of membership authorization and access control management while achieving great performance in terms of transactions per second (TPS).

The KRAKEN project architecture implements some of these key concepts which are briefly explained below as they are crucial for the understanding of the abstract layer of the architecture, i.e. the policies governing relationships between actors in the marketplace and therefore it's actual implementation

### 3.3.1 Organization

Every entity inside the HLF network is an organization and every member making transactions must belong to one. Each organization can set up roles for their members, such as “admin” or “peer”, not only to assign tasks and credentials accordingly but also to enforce, in the case of KRAKEN, access control policies. Technically, an organization is represented by a folder containing encrypted material called Membership Service Provider (MSP). The MSP is the means by which validation is performed when organizations or members transact. Lastly, organizations form consortiums, set consensus policies, and decide on the network configuration.

### 3.3.2 Certificate Authority (CA)

A Certificate Authority is an entity issuing certificates to authenticate organizations or users so they can perform actions inside the blockchain network. In essence, CAs sign Certificate Signing Requests (CSR) submitted by members who have been first registered by an admin. Also, an organization typically has a dedicated CA which is used to identify their own members. Other than that, they are responsible for checking if a certificate has expired, storing roles of members, etc.

### 3.3.3 Channel

A channel is a sub-network inside the system network which allows for private communication between organizations and organizations can specify their own rules for how the channel will be operated. Every channel may be created for a dedicated purpose and run specific smart contracts, depending on the use case. Also, each channel maintains a separate ledger.

### 3.3.4 Ledger

A ledger is a journal recording the history of transactions. The HLF ledger consists of the blockchain, which is an immutable transaction log, and a database called World State.

### 3.3.5 World State

The World State is a database keeping the current (latest) values for all the keys on the blockchain. The World State can be directly determined by reviewing the ledger and getting the latest state. The need for this database is to support efficient chaincode operations in terms of performance.

### 3.3.6 Chaincode

A chaincode is a package containing multiple smart contracts. Access control policies can be enforced inside the chaincode to ensure the eligibility of an entity performing read and write operations on the ledger.

### 3.3.7 Peer

Peers are nodes responsible for storing ledgers, running chaincodes, and endorsing transactions. When a transaction is submitted to the network, it is firstly handled by peers who execute the transaction proposal and endorse it if it produces a valid outcome. The endorsements must satisfy the channel policy for the transaction to be considered valid and subsequently get included in a block. After the endorsements are collected, the proposal is passed to the orderers.

### 3.3.8 Orderer

Orderers are nodes who packs transactions in an order and includes them into blocks after validating that they satisfy the channel policies. In essence, the process of executing and validating transactions is split, in Fabric, to reduce the work done by individual nodes and thus achieving greater performance and TPS. When blocks are created, orderers disseminate them to peers who in turn append them to the chain-code and filtering algorithm. Upon selection to browse the product catalogue, the application fetches the user's credentials from the database. The purposes property from the user credentials is checked against the products' purposes and filters all the products whose purposes do not match the user's credentials. In this manner, the data marketplace offers views that are relevant to individual users/product buyers.

### 3.3.9 Chaincode (Smart Contracts)

The fundamental objects governing data access permissions in KRAKEN are Smart Contracts (SCs) which handle and orchestrate compliance between users and products. The core operation in general terms needed to be secure, transparent, and immutable is the eligibility of data exchange between

participants. The diagram below illustrates relationships between users, data products and data access permissions as mediated by the chaincode.

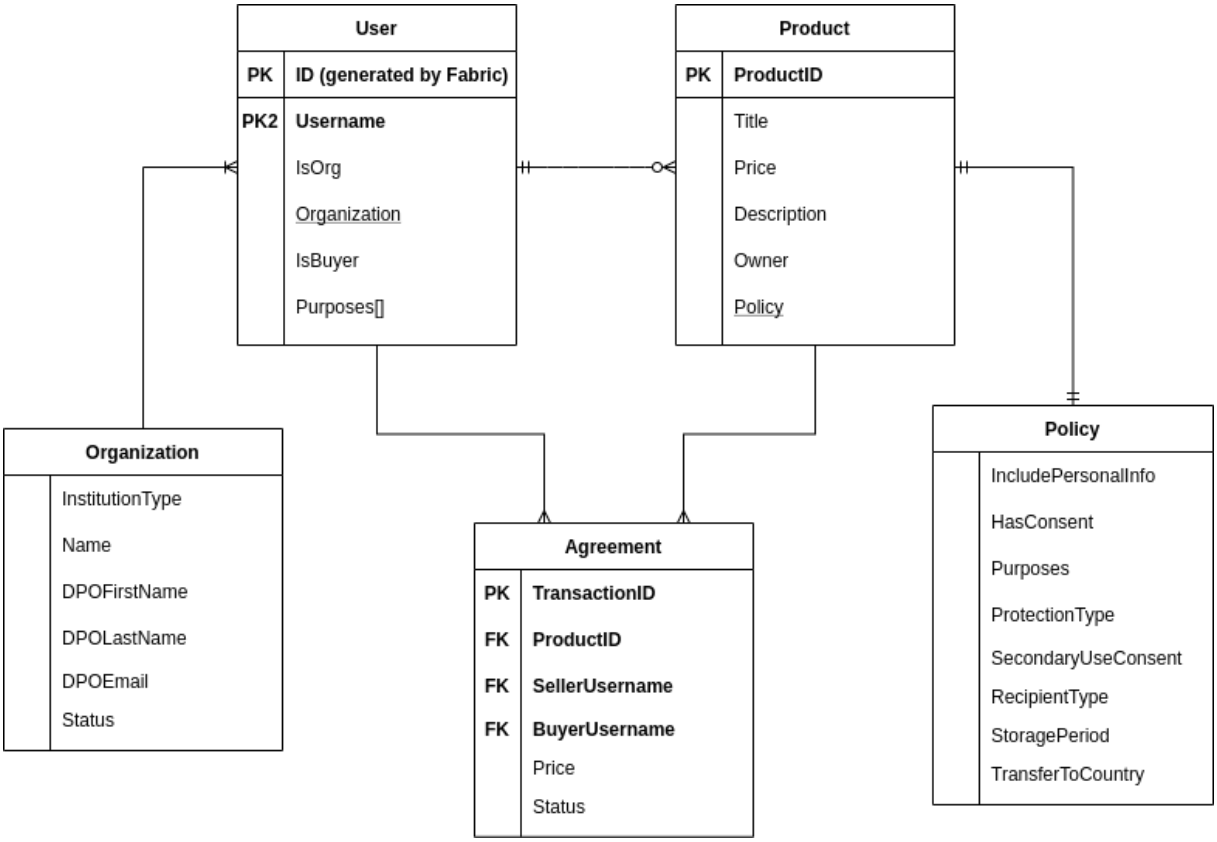


**Figure 3: Users, data access policies and organizations**

## 3.4 Access Control

The diagram above illustrates one of the two custom access policies allowed by Hyperledger Fabric in one of its layers, the chain code. An additional layer is provided by the channel configuration, where organizations can define how each member is authorized to perform various actions or have access to Fabric's resources. For example, in a consortium of organizations with organizations A, B and C, it is possible to set that the majority of organizations must sign a chaincode update transaction to be considered eligible. Or alternatively that A can be thought of as an owner and define that only A can update the chaincode. These policies have been set for KRAKEN and agreed upon with LYN, TEX, ATOS and ICERT as a prerequisite to deploy the data access layer, which, once deployed, enforces them by design.

### 3.4.1 Data access control layer policies

Before deployment in order to configure the Hyperledger Fabric policies, the organizations must first decide how the channel will be governed. The initial Lynkeus-TEX consortium, with the approval of the other organizations, has decided to follow the policies defined in the following table.

| Org (MSP) | Channel Update | Endorse |
|---|---|---|
| Lynkeus | required | required |

| | | |
|---|---|---|
| LynkeusUsers | | |
| TEX | required | required |
| TEXUsers | | |
| Other (non user) | required majority (including LYN,TEX) | optional |

**Table 1: Blockchain policies**

The chaincode layer, on the other hand is where data products access control is implemented via attributes. In any smart contract call, the contract can have access to the caller's certificate, making it possible to make decisions based on the organization that the caller is a member of. For example, data affecting a user registered in organization A can be set to have an endorsement policy requiring that A must endorse the transaction. As a practical example a data seller acting on behalf of an organization such as a hospital or a university on the marketplace, obtains in this way valid permission to sell access to the organization's data levering this mechanism.

### 3.4.2 Overall System Design

The system consists mainly of three components: network, application, and smart contracts. The network acts as a base layer to deploy the marketplace, while the application holds the back-end logic of the software stack. Essentially, the application handles user requests and database management and acts as middleware for the user-blockchain interaction as depicted in figure below

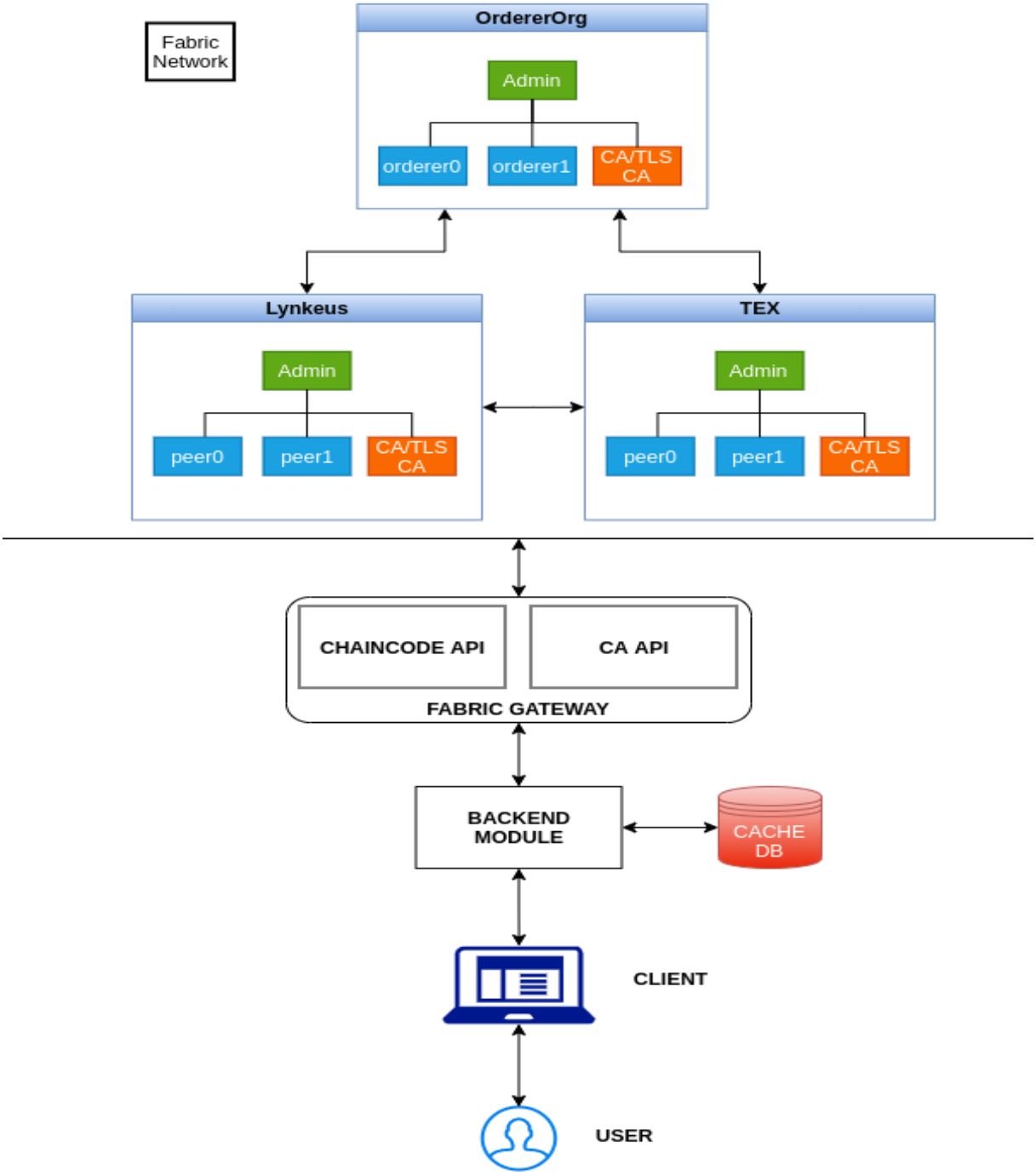


**Figure 4: Blockchain system design**

In order to maintain security and limit exposure, nodes are exposed only to the rest of the network and the application. As shown above, there is no direct connection between the user and the blockchain network. Consequently, all requests are passed to the back-end and forwarded to the relevant nodes.

The first challenge to be addressed with this approach is enforcing decentralization. In the standard implementation, the Fabric SDK both acts as a client and signs transactions on behalf of the user. This

approach assumes that crypto material is centralized, or that the application has access to a user's secret key. Therefore, since transactions are directed from the application, the user must have already signed the transactions with their secret key and only use the application as a client. To address this issue, we have designed a solution where the user, as the only holder of their private key, signs a transaction manually and forwards the transaction to the application. This way the network maintains trust, security, and privacy, while the user can safely assess that they and only they can alter their protected data, as explained below.

### 3.4.3 Network Architecture

The decentralized network, built on top of Hyperledger Fabric, aims to be the pillar of the whole marketplace infrastructure. The current pilot deployment consists of 2 peer organizations, Lynkeus and TEX, that represent the initial consortium. Also, an additional organization which we name OrdererOrg has the role of the Ordering Service in the network. OrdererOrg is not a distinct organization but is represented as such for the sake of clarity and role separation and will be governed by both Lynkeus and TEX. The organizations are joined to a channel in which a chaincode is deployed that handles the user and data logic. The network architecture is shown in the figure below.

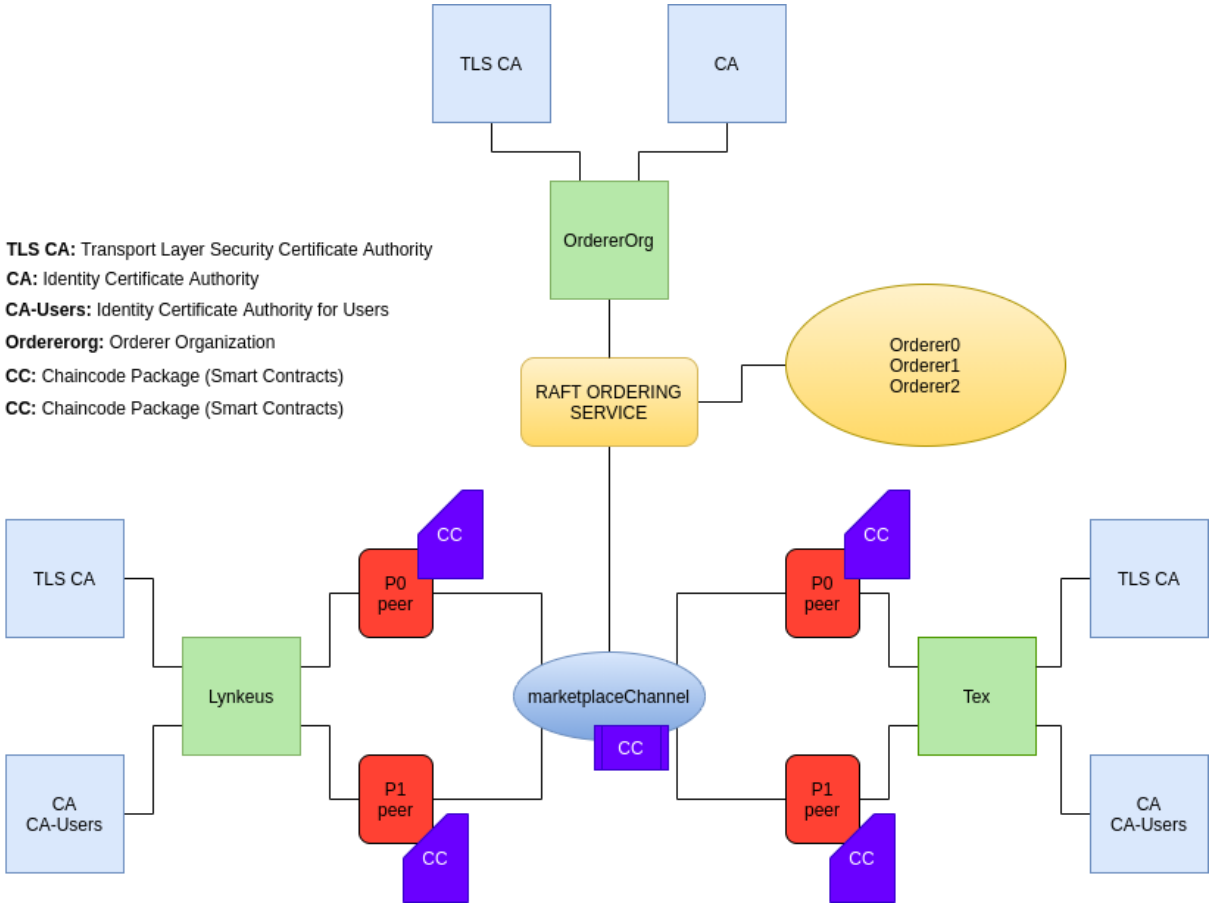


**Figure 5 Network Architecture**

#### 3.4.3.1 Components

Each organization comprises the following components:

- 2 Identity CAs
- TLS CA
- Organization's MSP
- Peers/Orderers

The **identity CA** is used as the root certificate authority which issues enrollment certificates. These certificates form the identities of all the nodes and users in the network and guarantee that they are approved by a consortium member. Moreover, each certificate holds information regarding the role of identities. These different types of roles are called Node OUs (Organizational Units) and currently, the following four roles are accepted on HLF: admin, peer, orderer, client. Using this role distinction design **we are able to create special access control policies and separate the network's actors**.

Each CA server deploys 2 distinct CAs, one for the members of the organization and one for the users that will be registered. The reason for this is not only to differentiate the root of trust for the members and the users and set specific certificate parameters for these two, but also to simplify the process of setting access policies within the channel configuration.

The **Trasport Layer Security CA** is required to enforce security against potential distributed network attacks. This CA is always deployed first and provides the certificate for all the communications between the nodes in the network. All nodes are deployed with two-way (server and client) authentication, adding an extra layer of security.

Next, we have the most important component of an organization, the **organization's MSP (Membership Service Provider)**, which forms its identity and allows for participation in the channel. Under each MSP, all nodes and user transactions are verified against the Ordering Service where it checks two conditions:

1. The transactor is a member of an organization joined in the channel.
2. The transactor is eligible to perform the intended action based on the channel's policies.

Both the TLS and Identity root certificates are stored in the channel configuration, so any member of the channel can message and authenticate messages and actions of other members.

Each peer organization hosts 2 peers in the initial deployment. While an organization can be a member of the channel, it is not necessary to host peers. It is important for organizations that endorse transactions based on the channel policy to have peers joined to the channel. For that reason, each of the initial organizations deploys 2 peers for the role of endorsing transactions. In the following sections, we will present the channel policies set for the most relevant resources.

Lastly, the Orderer organization, which deploys the ordering service, hosts 3 orderers. Being responsible for the block dissemination to peers, if the ordering service does not form a quorum, the network stops producing blocks and thus transactions cannot be committed. In our case, 3 orderers provide fault tolerance of 1 node.

#### 3.4.3.2 Deployment

Each of the components mentioned above represents a different physical or virtual node. In our implementation, each node is deployed as a docker container inside a Virtual Machine (VM) hosted in cloud services. The TLS CA is only used for the enrollment of nodes, so after the initial enrollment, it will be down as long as another node will not be joined to the network. Thus, we have used one VM for both of the CAs for better resource management. The individual network nodes are depicted in the following figure.

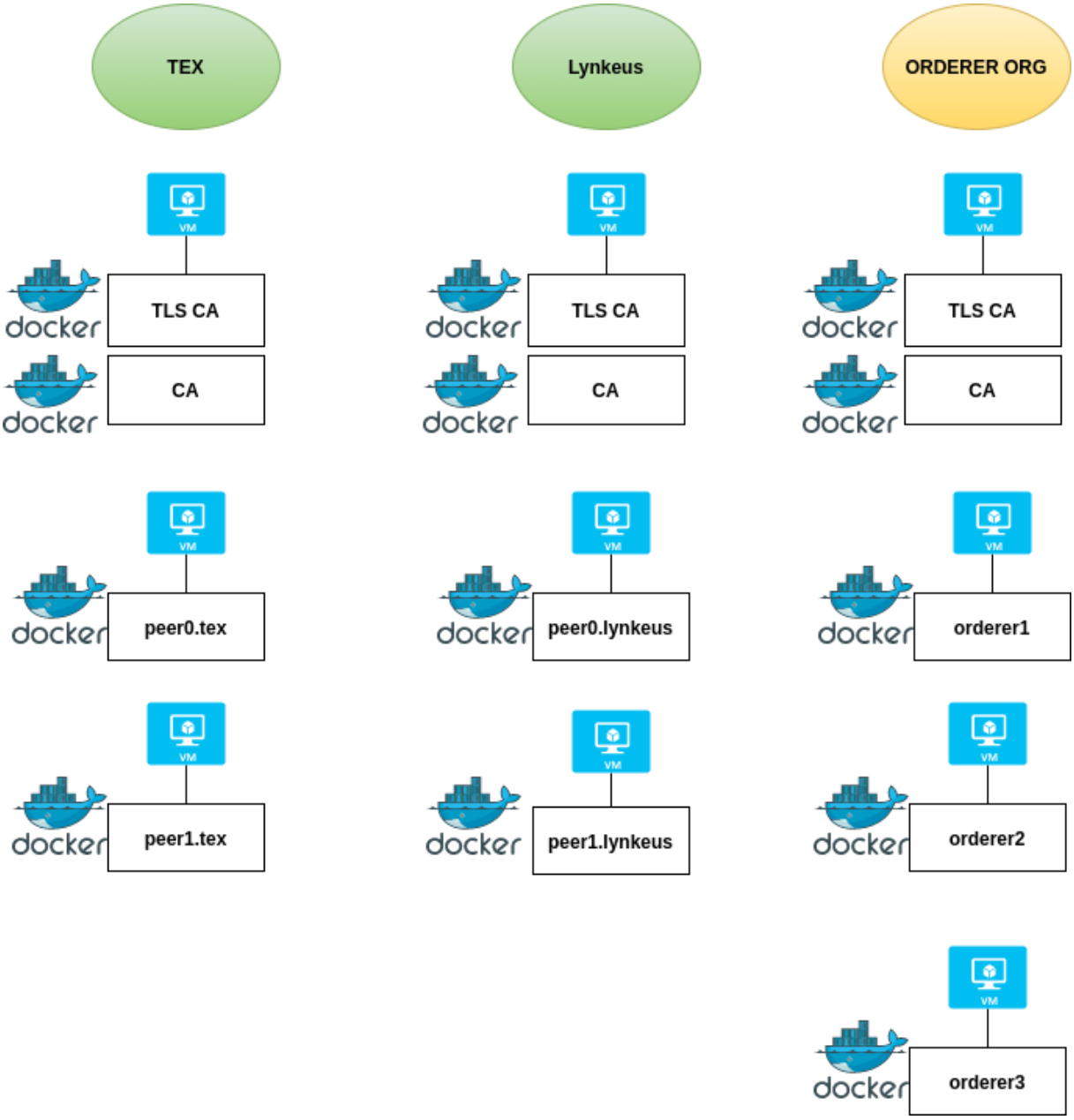


**Figure 6: KRAKEN HLF deployment architecture**

### 3.4.4 Filtering Algorithm

A user has the ability to browse the data marketplace and select data products to purchase. However, since buyers have predefined purposes and other preconditions (ex. country of residence and processing, age etc) for purchasing data products, it is essential to present only the products that the user is *eligible* to purchase. For this reason, we have implemented a filtering algorithm. Upon selection to browse the product catalogue, the application fetches the user's credentials from the database. The purposes property, along with other criteria, from the user credentials is checked against the products' permissions and filters out all the products whose access rights do not match the user's credentials. In this manner, the data marketplace offers views that are relevant to individual users/product buyers.

The Data Catalogue Service Certificate (SC) provides Create, Read, Update, Delete (CRUD) functionality for the products of the users. In addition to the basic operations, this function also invokes the Agreements contract when the user requests to buy a product. Each product is associated with a

strictly selected policy which in turn defines its lifecycle. The seller inserts the purposes that allow for use by another user. Although a user as a buyer has stated reasons for buying in their credentials, it is mandatory to state the reason for buying at each individual product they wish to buy, which is then handled by the Agreements contract. The Agreements SC is invoked on the buy product operation to decide if a user is eligible to access a specific product by matching the product's policy with the user's reason for buying. Specifically, the buyer's given use case is legally bound and recorded permanently on the ledger. If the agreement results are eligible, a transaction called agreement is stored on the ledger stating that the buyer has the authority to buy this product. Lastly, functionality is provided that allows for updating this transaction status via the application in the subsequent steps that follow the data lifecycle.

#### 3.4.4.1 Cache Database

The Cache Database is an off-chain solution to provide faster performance as well as greater flexibility for query handling. Since the platform can be used for data analytics, it is important to build a viable solution that allows for seamless additions in the future, further enhancing the platform's functionality and value. Our Cache Database is implemented as a MongoDB instance which replicates the data on the ledger via block event listening. As most of the blockchain platforms with smart contracts, HLF allows setting events inside the contracts to notify applications of updates on the ledger. Thus, a block listener updates the database with the data associated with the event and is also used to provide updates on the client side of the application. Additionally, our cache implementation allows for the reconstruction of the database from block 0 instantly without facing issues such as race conditions or data mismatches. Due to the asynchronous nature of such functionalities, it is essential to keep the data synchronized and inserted in the DB in the correct order. In order to accomplish this, we have created a mutual exclusion (mutex) implemented with queues on every user key which locks every time there is an ongoing operation on the DB. The key is unlocked once the DB finishes the operation.

The last thing to note is that the Cache Database is only updated via the block listener and exposes an API for querying operations.

### 3.4.5 Application-level integration

#### 3.4.5.1 User registration and enrollment flow

During the registration and enrollment process, a user must be guaranteed they are the only party that has had access to their secret key. By default, the API exposed by Fabric uses a wrapper for this process in which a client connects directly to the network, keys are generated locally, a Certificate Signing Request (CSR) is created and proceeded to the CA where it is validated. Since the architecture of this platform prohibits users from connecting directly to the network, we have split this process to ensure that the application does not have access to the user's secret key. To achieve this requirement, we have manually implemented the key generation and the CSR on the client side from where it is forwarded to the backend and then to the relevant CA. The registration process is made clear on the sequence diagram. Note that, in Fabric, registration and enrollment are two different processes but we are referring to the overall implementation of a user being a member of the network by getting a certificate.

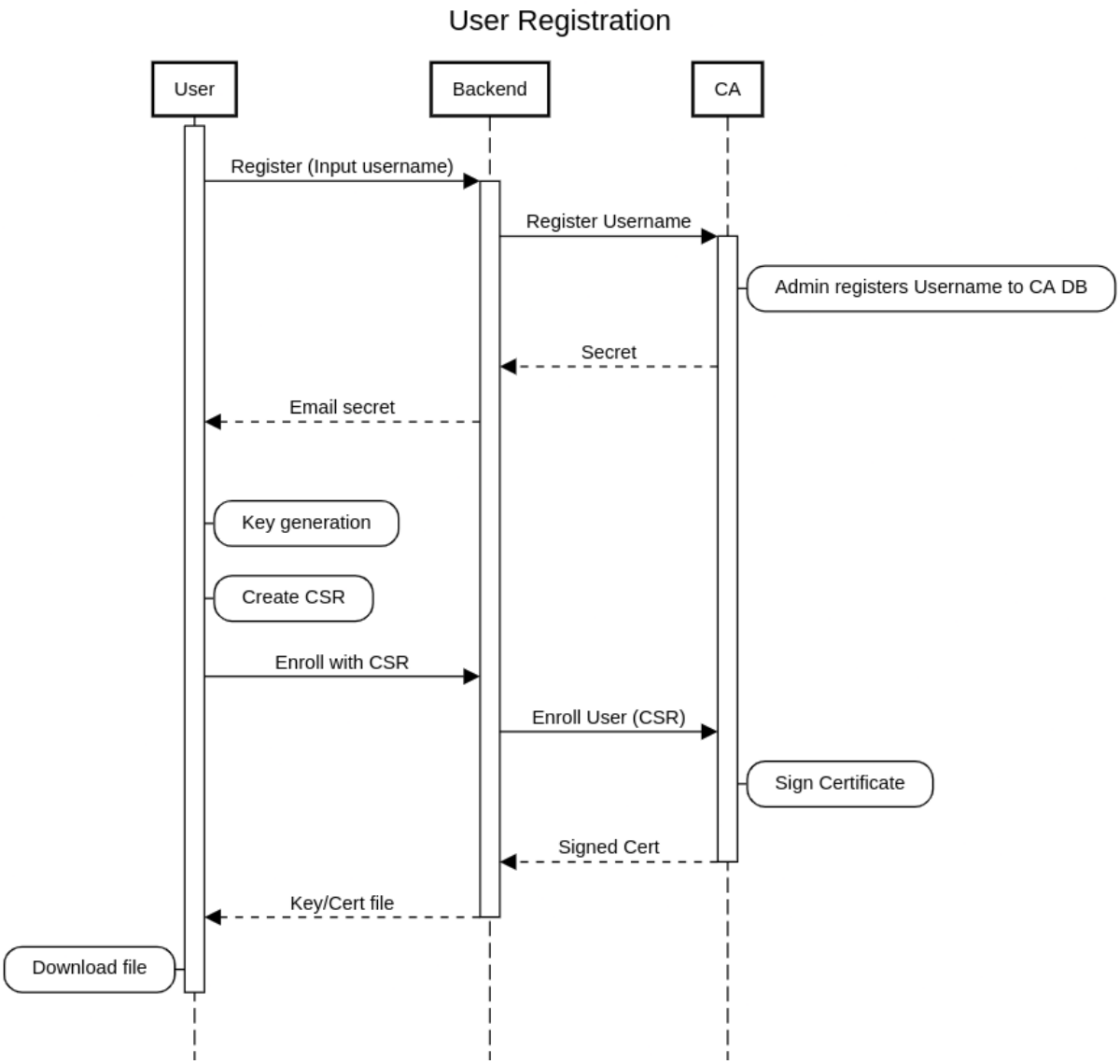


**Figure 7: User registration flow**

### 3.4.5.2 User transaction flow

In the same manner, as the registration and enrollment process, the SDK wrapper reads the signer's key imported from the file system and uses it to sign transactions and broadcast them to the network. Since the user communicates via the application, transactions must be signed locally on the client side and handled by the application, consequently preserving the privacy of their crypto material.

Essentially, the tasks of Fabric's transaction flow must be split. The sequence of the transaction lifecycle is:

1. The user imports the secret key on the browser where they sign the initial proposal which contains the transaction willing to make, such as creating a new product.
2. The transaction is forwarded to the peers, which validate and endorse the proposal.
3. The endorsed proposal is returned to the user who signs a commit proposal.
4. The commit proposal is forwarded to the orderers who in turn validate the policies and append the transaction to a block.
5. The block is disseminated to peers, added to the ledger, and the application handles the response.

The offline-signing transaction flow is also illustrated on the sequence diagram below.

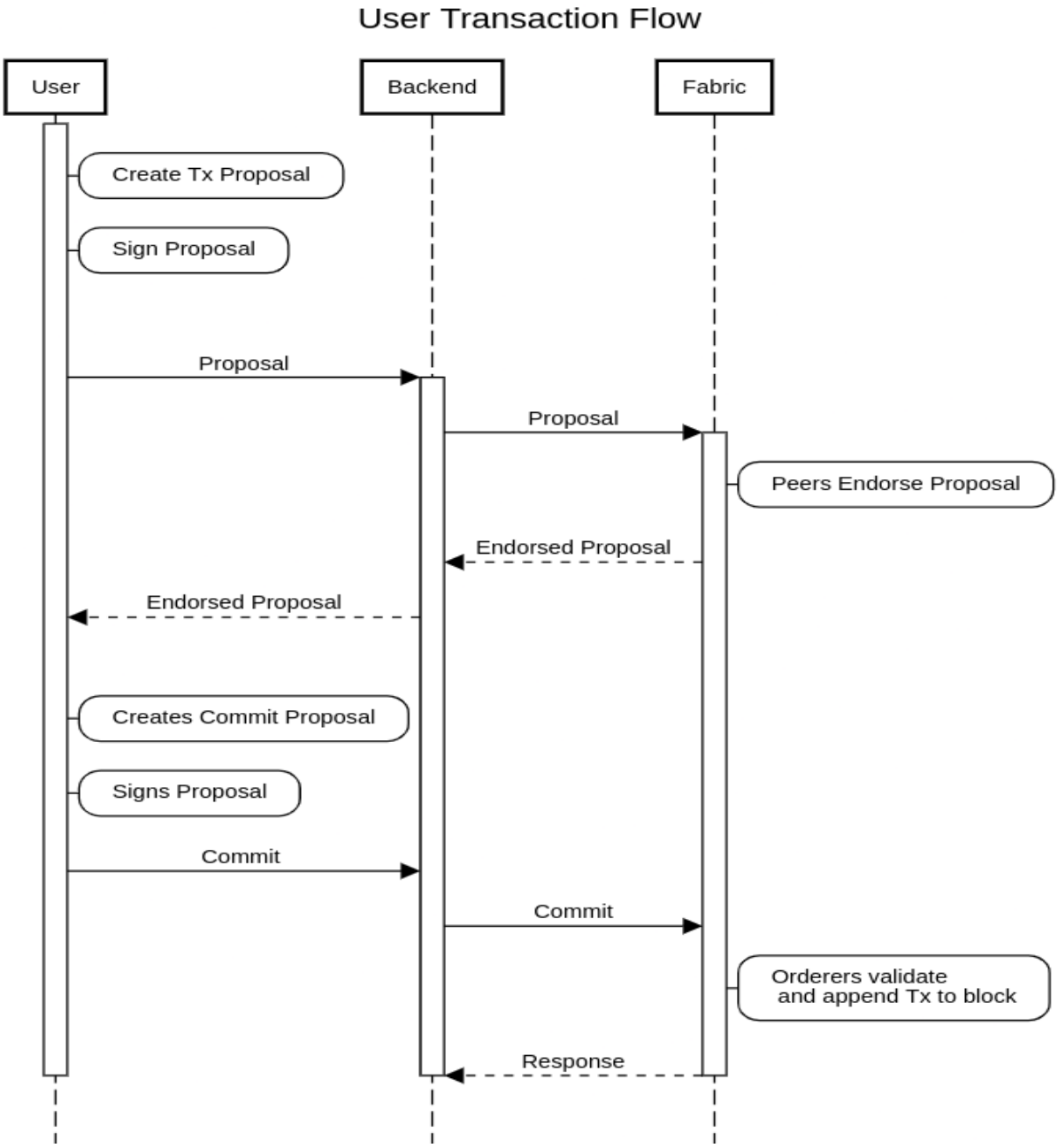


**Figure 8: User transaction flow**

#### 3.4.5.3 Installation APIs

Regarding the setting up of the network and the organizations, we have created APIs that simplify the most essential processes of a peer and a CA client. These APIs can be used by a network operator to set up an organization, join a peer to channel, install chaincodes, manage the CAs, and so on.

| CA Client API Tasks |
|---|
| Register user to TLS CA / CA |
| Enroll user |

| |
|---|
| Re-Enroll user |
| View list of Identities in CA |
| Setup and launch CA Server |
| Setup Organization MSP |

**Table 2: CA API Tasks**

| Peer API Tasks | |
|---|---|
| Creates a channel transaction from profile config | Install chaincode to peer |
| Submit channel transaction to orderer | Query installed chaincodes on peer |
| Joins a peer to channel | Approve chaincode as Org |
| Create and submit anchor peer update transaction | Query approved chaincodes on channel |
| List channels a peer has joined | Check commit readiness of chaincode |
| Package a chaincode | Commit chaincode definition to channel |
| Query committed chaincodes on channel | Fetch configuration of channel |
| Create an update transaction to add an organization | Sign configuration transaction as organization |
| Start a node, peer/orderer | |

**Table 3: Peer tasks**

## 3.5 Data access modalities

As the data access permission layer allows compliant data access requests to be executed, permissions are passed to the data transaction player which allows three types of data access.

1. **Batch data transfer through directory sharing.**
   Data storage happens always outside of the platform, which provides only encrypted local addresses to the buyer at the time of purchase. Sellers data are also encrypted and the platform, without storing or in other ways coming into contact with it, allows the buyer to access the encryption key at the time of purchase. The buyer therefore can access the local repository and fetch the data product for the intended use and contractually agreed upon period.
2. **Transfer of encrypted streaming data**
   Here the TEX infrastructure channels data streams using its distributed network that connects data sources with subscribers to data streams that have been purchased on the marketplace. Streams are activated and kept open under the same data access mechanisms as in the case of batch data access. This use case is applied to a subset of biomedical data only, i.e. data from wearable and monitoring devices, due to their semi-continuous nature as opposed to static healthcare data (ex. electronic medical records) and educational data.
1. **Secure Multi Party Computation**
   Local data assets, in this case, are accessed only by the distributed querying mechanisms (SMPC) which retrieves precomputed query results to the user without exposing any of the

underlying data. The permissioning framework and correspondent implementation in the Lynkeus blockchain is currently being designed for this data access modality. SMPC is also used to share encryption keys for the transfer of encrypted batch data.

# 4 Transaction management, the Streamr data infrastructure

Streamr is a project that aims to realise a decentralised worldwide network for real time data sharing. In its current state, Streamr is not yet fully decentralised, but it's already a Peer-to-Peer publish-subscribe network for real time data transfer. It works with IoT devices, applications and anything with an internet connection that can run the Streamr client software.

The Network is formed by a set of broker nodes. These nodes are intended to be installed on always-on machines and connected to other nodes to route the traffic. The governance of the Network is performed by a smart contract on the Ethereum blockchain. On this smart contract are saved all the information regarding coordination, permissioning and access control of data streams. The actual transfer of data happens off chain on the Streamr Network that benefits from the "network effect" as with the increasing number of nodes, the scalability increases as well.

KRAKEN integrates Streamr in a way that the publishing process of data providers filters the subscribing data consumers on the Streamr Network so that only the data consumers that are registered on the KRAKEN marketplace and are eligible to subscribe to the stream are able to gain access to the data streams.

This mechanism is summarized in the following image:

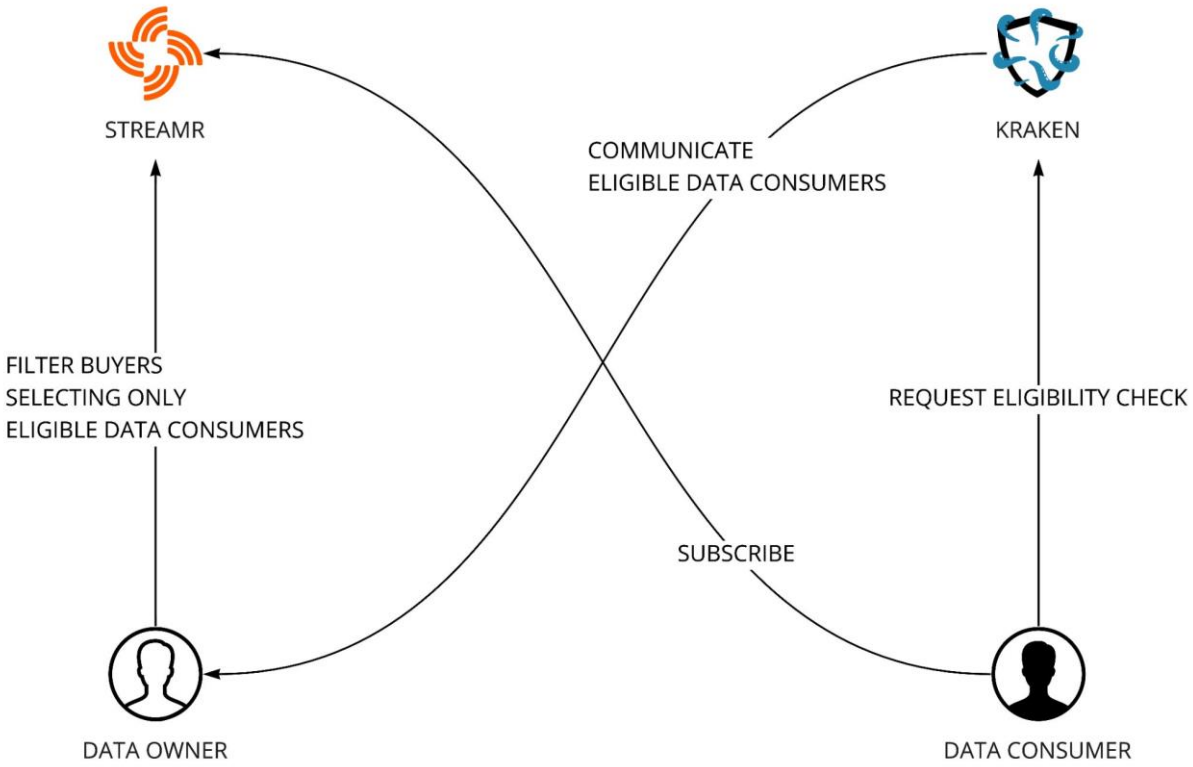


**Figure 9 Transaction Management**

## 4.1 Batch data

The Encrypted Batch Data Product use-case enables data transactions that allow sellers of data access to give eligible uses access to their datasets whilst preventing any other actor outside of the transaction, including the KRAKEN marketplace itself, from accessing the data. This is achieved by exploiting the smart contracts that are implemented on a permissioned blockchain (Hyperledger Fabric, as illustrate in section 2.3) and a Multi-Party Computation (MPC) network. Security features of this mechanisms are explained more in detail in the Data Protection Layer section.

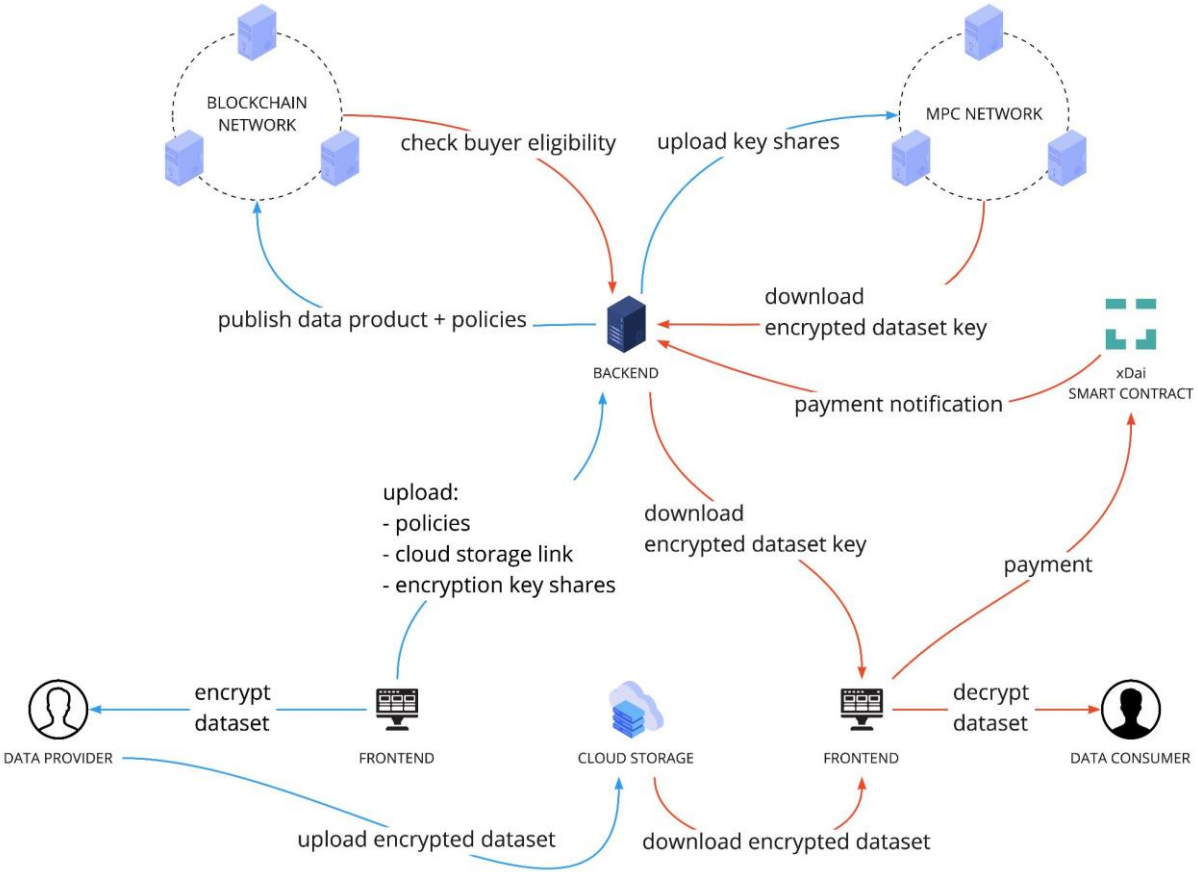


**Figure 10 Batch data transaction**

The data flow begins with the data provider, who commences the Data Product publication workflow in the marketplace UI by entering the relevant information about the Data Product and the policies that dictate the eligibility of data consumers to access the data set. Once all of the relevant information has been entered in the marketplace UI the data publication process commences. This process is described below and begins with a data provider who has a data set stored in their own local storage that needs to be published on the marketplace.

1. The Data provider provides the dataset to the KRAKEN frontend (a web page that is running at that moment on the user's laptop).
2. The KRAKEN frontend generates a random encryption key, encrypts the data sets with that key and gives the data set back to the user. All this happens locally without any requests sent to an external server.
3. The user is requested to upload this encrypted dataset on a cloud system and make it publicly accessible with a link.
4. The user provides the link to the KRAKEN frontend.
5. The KRAKEN frontend splits the key previously generated in as many shares as the number of MPC nodes in the MPC network. This operation prevents anybody without a sufficient number of shares from reconstructing the original key.
6. The KRAKEN frontend sends all of the gathered information to the KRAKEN backend.
7. The KRAKEN backend stores the metadata of the Data Product, instructs the blockchain with the new Data Product together with the policies and sends the key shares to the MPC network nodes.

In the figure above, the data publication process described above is portrayed using blue arrows.

On completion of the process described above, the KRAKEN marketplace has a Data Product available for access and purchase by data consumers. The Data Product access and purchase process is described below.

1. The Data Consumer browses the catalog of data products and chooses a Data Product.
2. The frontend locally generates a keypair and requests to the kraken backend the Data Product access purchase by that specific Data Consumer including the keypair's public key.
3. The Consumer's identities and other Verifiable Credentials, ex. their institutional affiliation are checked against the policies set up by the Data Provider on the blockchain.
4. Once the frontend receives the Data Consumer eligibility confirmation, the user performs the payment (still exploiting the kraken frontend).
5. Once the backend is notified of the payment, it sends a request to the MPC network to provide the Data Product corresponding dataset key encrypted with the Data Consumer public key.
6. The KRAKEN frontend receives the key and the dataset link, decrypts the key using the keypair's private key, with the obtained key it decrypts the datasets and provides it to the Data Consumer.

In the Figure above, the data access and purchase process described above is portrayed using red arrows.

On completion of the process described above, the data consumer has obtained the data set from the data provider. Throughout the above-described process, the status of the transaction is tracked by the backend, which updates the blockchain with every change of status. Specifically:

- When an eligible data consumer requests a Data Product, the transaction's status on the blockchain is updated with this information.
- When the user has performed the payment and the backend has received a notification from the smart contract on the xDai network, a stable payments blockchain designed for fast and inexpensive transactions (xdaichain.com) leveraged by TEX in their Streamr platform, that this has been completed, the transaction's status on the blockchain is updated.
- When an eligible Data Consumer that has already paid for access to the Data Product requests the Data Product's corresponding dataset, if the process of providing the link and decryption key is successful, the transaction's status on the blockchain is updated.

## 4.2 Biomedical streaming data

As described above, this use case applies only to a subset of biomedical data from wearable and monitoring device in particular, in view of the fact that educational and other types of biomedical data have inherently a static nature. The encrypted healthcare streaming data use case allows data providers to give data consumers access to their data streams in a way that prevents any other actor, including the KRAKEN marketplace itself, from having access to the Data Product. This is achieved by exploiting the smart contracts that are run on the Data access permission blockchain, the Streamr Network and xDai.

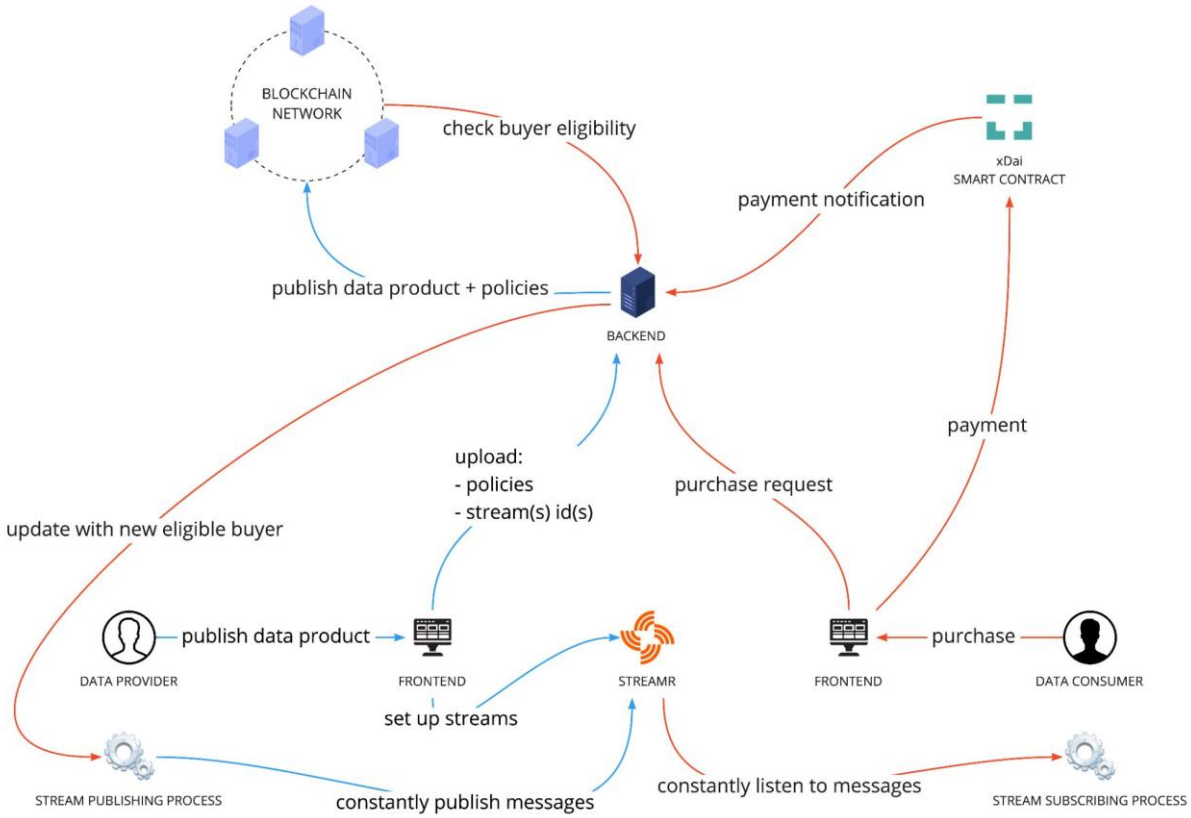


**Figure 11 Biomedical data transactions**

The management of educational data is explained below in the context of the SSI and Verified Credentialing (VC) because of its deep dependency from those systems, in light of the fact that educational data are effectively treated as VCs.

# 5 KRAKEN mobile application

## 5.1 Application architecture

The KRAKEN Platform will utilise one mobile app serving multiple purposes:

1 User creation and authentication with SSI
2 Simplified management of data products
3 Monitoring of data products uses
4 Exercising of user rights in compliance with the GDPR (ex. Right to be forgotten) and policies management

The app's architecture will therefore include an SSI SDK which is currently being extended with a REACT native module to connect user facing functionalities to the underlying SSI components relying on the Aries Framework (see below).

The mobile app has a particularly critical role in the implementation of the educational data use case, in which the app serves as local storage for not only user identities (DID) but also of their verified credentials, among which are data related to their academic grades and achievements as explained in section 6. Contrary to the biomedical pilot where raw, individual data, weather in batch or streaming forms, constitute the actual data product, in the educational pilot verifiable credentials are exchanged for value, realizing a structurally different use case and corresponding architecture as explained in section 6.

# 6 The Data protection layer

## 6.1 SMPC internal architecture and privacy features

The SMPC security infrastructure has been discussed in depth in other deliverables**.** For further information about the envisioned cryptographic mechanisms that enable a data receiver to obtain privacy-preserving data-analytic results from data owners, and hence further information about the features of MPC, we refer to Deliverable 2.4 (Kraken intermediate technical design). Specifically Sub-Section 3.1.1 (State of the art SMPC software) within Section 3 (Cryptographic tools and analytic engine specifications). Furthermore, also in Deliverable 4.1 (Progress report on cryptographic protocols for privacy-preserving data markets and SSI systems) the usage and some properties of MPC are described. Specifically, in the contribution of T4.2's paper *Privacy-preserving Analytics for Data Markets using MPC* (Sub-Section 3.1) as well as T4.3's paper *Multi-Party Revocation in Sovrin: Performance through Distributed Trust* (Sub-Section 4.2).

Here we explain two application of SMPC in the KRAKEN marketplace, namely the sharing of encryption keys for the sharing of batch data and the distributed analytics use case, which is currently under definition.

## 6.2 Protected encryption key sharing for batch data

One of the central issues that the KRAKEN marketplace has to solve evolves around end-to-end secure sharing of data between data owners and data buyers. With the marketplace architecture we face multiple issues that are normally not encountered when transferring data securely between two communicating parties. The KRAKEN marketplace enables data owners to advertise their data on the marketplace such that data buyers can browse the offerings and buy the data at any time. Here we can already observe the first challenge for achieving end-to-end security: when using classical approaches, the data owner is required to be online when the buyer triggers the data exchange process. Indeed, if we consider a system where each buyer possesses a public key, the data owner only knows the target for encrypting their data when the buyer actually buys the data.

With the marketplace in the middle, one could of course tightly integrate the marketplace in the data exchange as trusted third party. In this case, however, the marketplace would have to manage the data owner's data in an unencrypted way. *This would on one side impose substantial legal requirements on the platform, but also drastically reduce the level of decentralization*. From a security point of view, the marketplace would be, indeed, an additional single point of failure. Compromising the marketplace would then potentially give an adversary access to all data. When employing techniques such as proxy re-encryption, we are able to secure the data from the marketplace. Yet again, the seller would need to be online to generate proxy re-encryption keys whenever a buyer is buying access to a dataset. Overall this means that either we have to put additional trust in the marketplace or require the data owner to be almost always online.

To address these issues and challenges, the KRAKEN marketplace design follows a distinct approach to provide both end-to-end security between seller and buyer while at the same time not requiring the sender to be online for the data exchange. For the following discussion, recall the KEM-DEM paradigm. For encrypting arbitrary data, one first samples a random key for a symmetric encryption scheme which is used to the encrypt the actual data. The symmetric key is then encapsulated using public key encryption scheme with respect to the receiver. Therefore, we can focus on the end-to-end secure transport of the symmetric key. The encrypted data can be stored on any cloud storage service and sharing of the URL can be managed by the KRAKEN marketplace.

We consider the following setup:

- The data owner which we call the sender and the data buyer called the receiver.

- A set of at least three multi-party computation nodes run by different parties. One of these nodes performs a slightly different tasks than the others and is called the aggregation node.

The idea is that the sender first performs a linear secret sharing of the key producing one share for each MPC node. Then, the shares are individually encrypted using a linearly homomorphic proxy re-encryption scheme for each of the nodes and sends the ciphertexts to the nodes. After this step the participation of the sender is no longer necessary. When the receiver requests access to the data and thus the key, the marketplace triggers the following process after verifying that the receiver satisfies some requirements set by the sender: the marketplace sends the public key of the receiver to the MPC nodes. The nodes re-encrypt their ciphertexts to the receiver. Then they send the ciphertexts to the aggregation node, which recomputes the original symmetric key in the encrypted domain. The so obtained ciphertext is forwarded to the receiver which is then able to decrypt the ciphertext.

Under the typical MPC assumption that at least one of the MPC nodes is not compromised, even if all other nodes collude, they are unable to recover the original secret key from their shares. Additionally, to give the receiver authenticity guarantees on the symmetric key, our design is accompanied by a linearly homomorphic signature scheme. Besides combining the individual ciphertexts, the aggregation nodes also combine accompanying signatures which allow the receiver to verify the symmetric key. Figure below depicts the sequence diagram of this protocol.

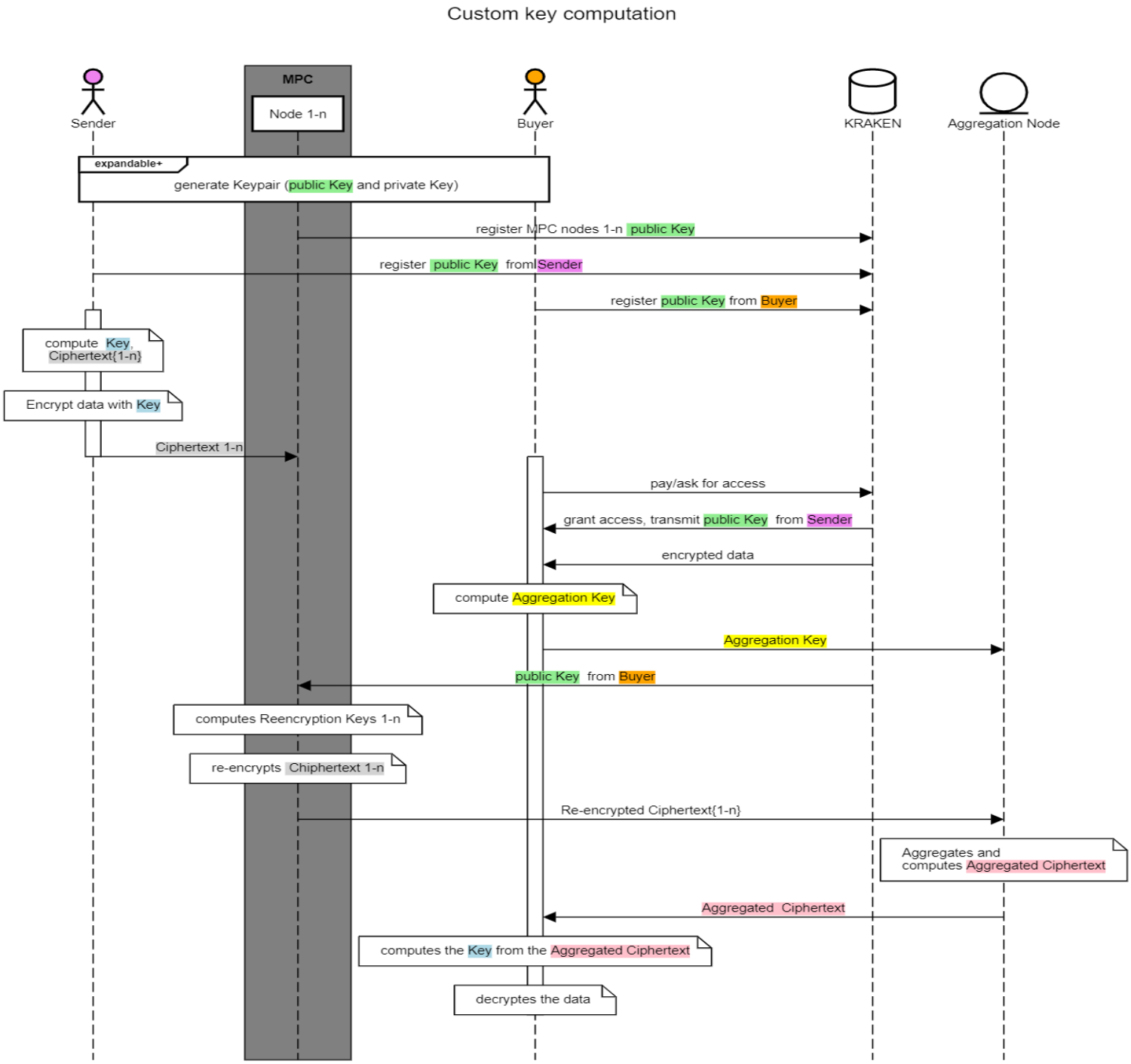


**Figure 12: Secure secret keys sharing**

### 6.2.1 Distributed analytics via SMPC

SMPC will be utilized to implement the third data access modality, i.e. the distribution of encrypted queries over multiple datasets which will not be transferred but rather interrogated in this fashion to retrieve query results to the user. The same underlying permissioning system will be utilized to support data access control in this specific modality. User workflows are currently being designed for data access buyers to select and instantiate queries, pick the target databases, execute payments and finally retrieve the information they seek. Designs for all this integrated functionalities are being developed and will be then implemented on time for the first platform release.

### 6.2.2 Protection of streaming data

The healthcare streaming data encryption exploits the Streamr streams encryption mechanism available when using encrypted streams. Specifically, this encryption is end-to-end. When a data consumer requests access to a data stream, the request is sent to the data provider. This request includes the Ethereum address of the data consumer and his public key. Once the data provider's publishing software has checked that the KRAKEN marketplace granted the permission to that specific Ethereum address, it shares with him the encryption key of the stream's messages (encrypting it with his public key).

In this way the stream of messages sent from the data provider to the data consumer can be known only by them. Even the KRAKEN marketplace or the Streamr network cannot access the data.

# 7 Self-Sovereign Identity

## 7.1 User Authentication

As detailed extensively in D2.4, Self-Sovereign Identity systems allow individuals to control and supervise how their digital identity is used or shared in a framework in which users can exist independently from the services they access, as opposed to traditional digital identities administered by centralized authorities like governments, corporations, or software platforms providers. As the next step toward user-centric this realizes interoperability across systems and environments and true user control.

As in all models of identity management, a digital identity requires **identifiers**: something that enables a person to be discovered and identified and which ensure the user is who he/she claims to be. In self-sovereign identity, identifiers do not need an intermediary. Rather, globally unique persistent identifiers, not relying on a centralized registration authority, are generated and/or registered cryptographically as **decentralized identifier** (DID). Most of DID methods use distributed ledger technology (DLT) or some other form of decentralized network.

Users are at the same time able to generate **credentials**, as personally identifying information or facts about themselves, including information asserted by other persons or entities. In such cases, those entities are considered **issuers**, while the user (owner) becomes the **holder** of that credential which may or may not relate to the subject of the credential (e.g. newborn baby birth certificate held by a parent). The third role that an entity can play in a SSI system is the **verifier**, able to request a credential from a user (holder) and will verify its validity.

An important pillar for the SSI is the **Distributed Ledger Technology** (DLT), typically built on a peer-to-peer networks using consensus algorithms that ensure replication across the nodes of that network. Together with DID and DLT, the third pillar on SSI are **verifiable credentials**, i.e. digital assertions made by a user (or entity) containing a set of information about itself or another entity, that can include a digital watermarking as a cryptographic verification of its content or a part of it (i.e. a **claim**).

A typical SSI architecture is presented in the following diagramFigure 13.

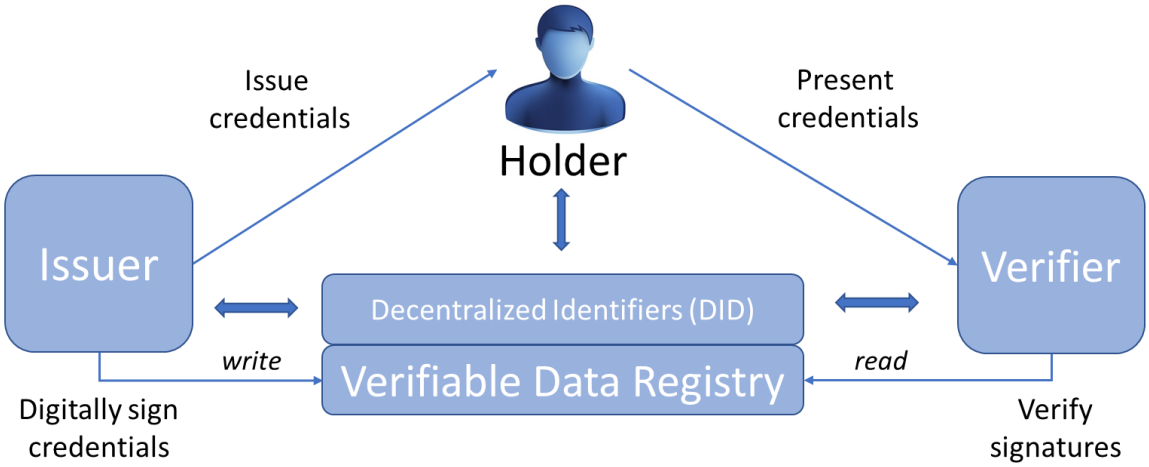


**Figure 13: SSI architecture**

The Verifiable Data Registry acts as a holder of the signatures of the issued credentials and supports the creation and verification of decentralized identifiers. This registry is based on blockchain technology.

A typical flow in these architectures is:

1. A user needs a credential and proceeds to ask the issuer for it.

2. The issuer requests the user to be identified and exchanges its DID
3. The issuer generates a credential for this DID, based on the user´s information, and record the signature of the credential generated in the blockchain.
4. Once the user receives the credential, she/he is able to present this credential to the verifier.
5. Verifier checks blockchain in order to validate the signature of the credential, to guarantees the authenticity of the data, without actually storing any personal data on the blockchain

### 7.1.1 The KRAKEN approach

KRAKEN leverages the above-mentioned approach in a way that is agnostic to the underlying blockchain technology giving the consortium freedom in developing a solution that will not be tied to a specific ledger implementation and consequently it will be easily integrated/adapted with any service.

The holder will use a mobile-based application, the KRAKEN app mentioned in section 4, to interact with the issuer and service provider, and to store the credentials, in order to present them in any service that requires them. This application contains an interface to connect with a credentials backup service in charge of making a backup of the cryptographic material used in the SSI. This way, if the holder needs to use another mobile device (or multiple) or in case their devices have been stolen, she/he will be able to restore all material needed to replicate the SSI.

As it was envisaged in D.7.2, KRAKEN would use eIDAS (electronic IDentification, Authentication and trust Services) for electronic identification and trust services in the European Single Market for the purpose of identifying the holder and will add this information as part of the generation of the Verifiable Credential. The Legal Identity Manager (LIM) is, on the other hand, the issuer that generates and sign this identity on the basis of the eIDAS information, provided by the holders through their national authentication services.

All this strictly aligns KRAKEN with the European Self-Sovereign Identity Framework (ESSIF: https://ec.europa.eu/cefdigital/wiki/pages/viewpage.action?pageId=262505734) .

## 7.2 Educational data exchange through Verifiable Credentials

These functions and respective integration come together in the context of KRAKEN's educational pilot in which users can export their academic information from a university system and provide access to various stakeholders such as other academic institutions, recruiters or employers. It's very important to notice here the *difference with the biomedical pilot* in which data are changed in either batch or streaming forms, while educational data are encoded and exchanged as verified credentials. The KRAKEN marketplace has been, in this view, directly integrated in the Gratz university information system. Key to this integration is the Hyperledger Aries framework in which every stakeholder operates an Aries agent including the university and the student provided with a wallet to store DID and VCs. In the future, other stakeholders like human resources agencies will be added to the demonstrator.

Agents interact with the information systems either by using REST APIs, or interfacing directly. In the current demonstrator, the university uses a central API service for all kinds of university-internal data exchanges like access to the student-information-system, library, and room reservation system. Since the agent used by the KRAKEN pilot is also integrated into the university API, the KRAKEN credential exporter frontend communicates with the agent using his API. Thus, the frontend needs only a single point of contract at the university for access to student data as well as communication with other Aries/KRAKEN agents. Additionally, this layer takes care of access control and ensures that students can only export their own data, which furthermore ensures the student's consent at all times.

To communicate with other agents, an Aries agent implements various communication protocols which use the student's DIDs to establish a connection with the student's KRAKEN app (typically a mobile wallet). To resolve the involved DIDs, the agent requires access to the corresponding DL, which

is realized by using DIF's sidetree protocol. After this step, the university and the student agent establish a direct communication using REST APIs. This connection is then used in the background to issue credentials directly to the student's wallet.

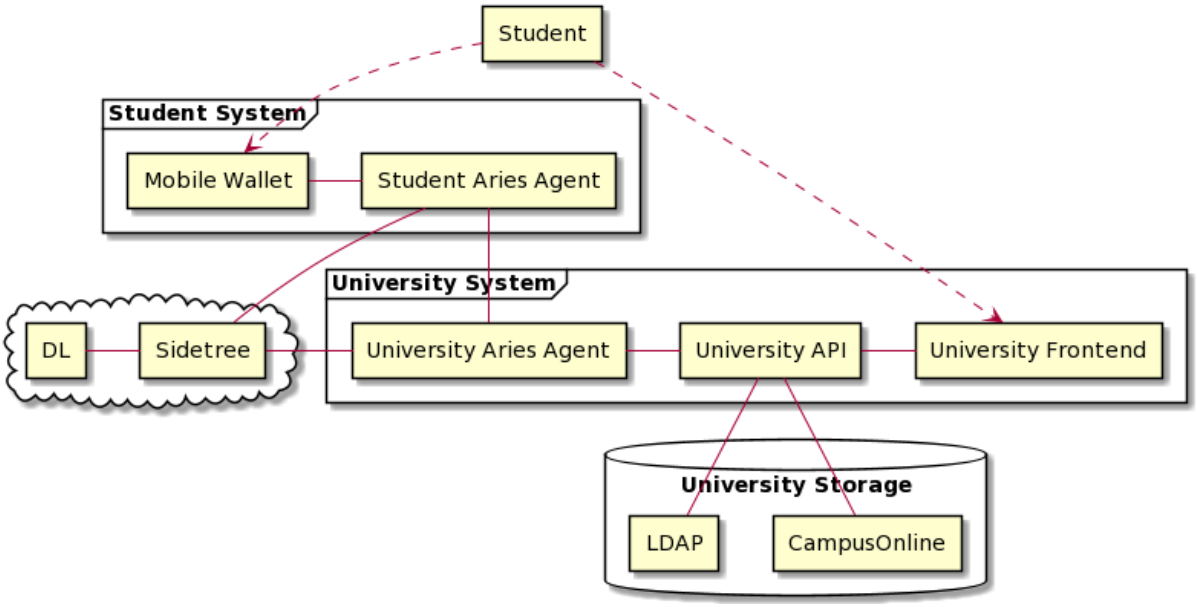


**Figure 14 Wallet and agent in the educational pilot**

### 7.2.1 Education data exchange, user workflow

To facilitate the integration of the KRAKEN system multiple modules, which have been discussed in detail in D2.2, have been developed to support the communication between issuers (universities) and holders (students), as well as the issuing and storing of VCs using the Hyperledger Aries framework. These also facilitate the integration of the educational data domain into the KRAKEN ecosystem hinging on the KRAKEN app as the main interface to the marketplace. Data stored by it can indeed be directly used within the marketplace. Education data are exported into the student's (mobile) wallet to then be exposed on the marketplace as downloadable assets or targets of distributed analytics with SMPC as the system takes care of the encryption of the data, so that no one except for the qualified buyer has access to the plaintext, not even the marketplace itself.

The diagram below gives a high-level overview of the dataflow of the analytical data product use case.

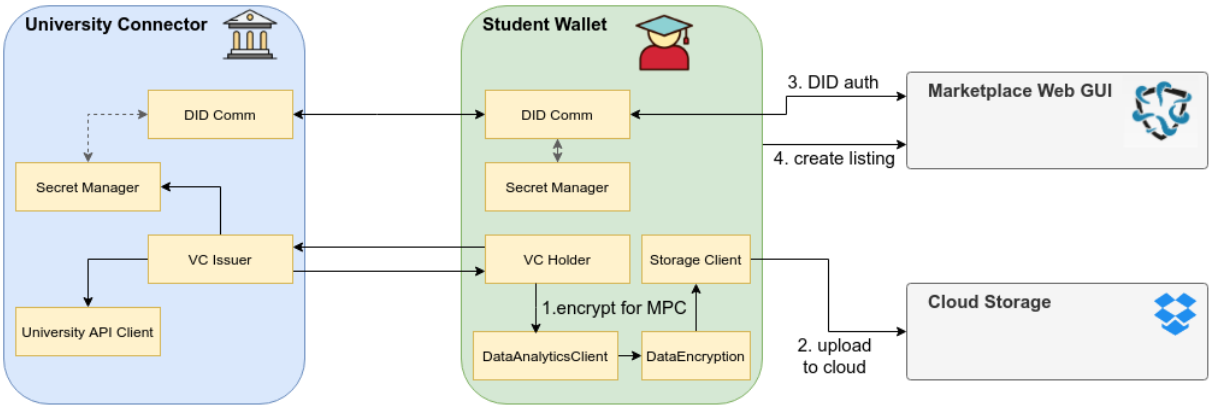


**Figure 15 Educational Pilot Architecture**

### 7.2.2 Application-level integration

Key to this architecture is a connector component which interacts with the (existing) university systems on one end and with the KRAKEN marketplace on the other and consists of a web interface which students can use to connect their SSI/KRAKEN app (see below) and perform a decentralized identifier (DID) handshake. Students use their university account, eIDAS login, or other E-ID (like ID Austria) to first login. Once the DID connection is established, students can request that their course certificates and graduation diplomas are issued in the form of VCs. The issuing is performed by the backend of the education connector and issued credentials are directly sent to the student wallet using the respective DID communication protocol. This means there is no need to scan multiple QR codes for each credential.

Since the process can only be initiated by the student themselves and requires their presence in the interaction process at all times, the explicit consent of the student to the data export is ensured. This is reinforced by an informative consent screen during the authentication process. Additionally, data is only transferred directly to the student's wallet and no third parties have access to the data at this stage.

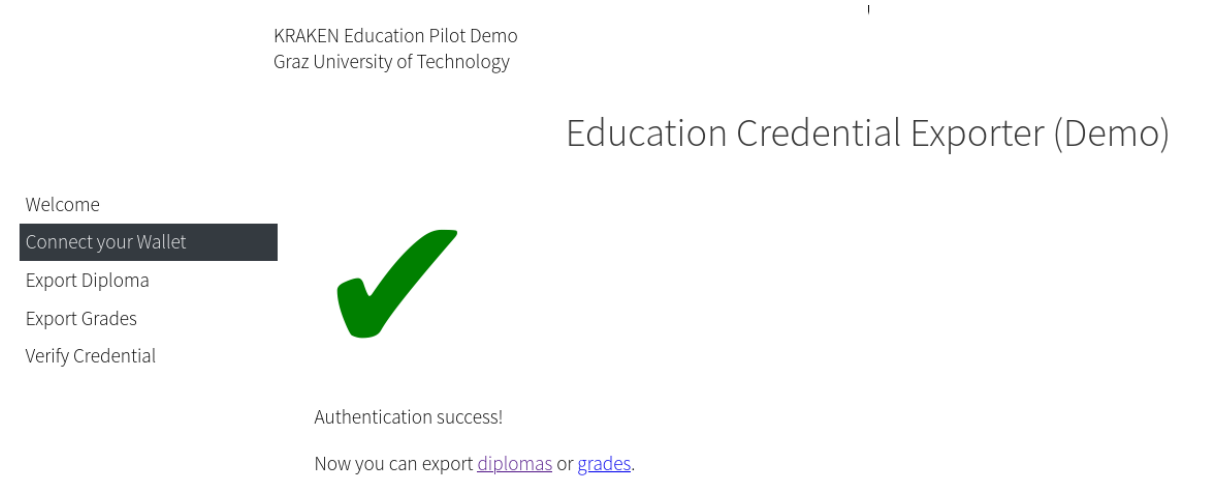


**Figure 16 The connector after a student connected their KRAKEN wallet**


**Figure 17 The Connector before a student exports a diploma credential to their wallet**

After exporting their credentials into the KRAKEN app, students can use the credentials in a self-sovereign way without further involvement of the university.

In addition to directly show credential to verifiers, students can use them to create data products on the KRAKEN marketplace. Doing so enables buyers from various fields and industries to buy access to

these credentials or a set of credentials or use them in statistical computations -- with the student's consent and with the marketplace taking care of rewards and payment processing.

To do so, a student uses the KRAKEN app to authenticate and prove their affiliation to the university. Afterwards they create a data product by providing some metadata about the credential, payment information, and the credential data itself. Further, the student defines access policies for their data like who can buy the data or in what computations the data can be used. The marketplace app takes care of the rest, encrypting the data and uploading it to a cloud storage.

# 8 Outstanding issues in decentralization

In a strictly decentralised system, correct functioning does not depend on any single entity (person or organisation) and operations, therefore, cannot be corrupted by targeting that entity. Attackers are forced to compromise the majority of the nodes in the entire network to enact malicious intents. In KRAKEN there are two decentralized systems: the Lynkeus Blockchain and the Multi Party Computation (MPC) network.

In the Lynkeus blockchain, the process of the eligibility check exploits smart contracts in the Hyperledger Fabric network, decentralizing the data access permission process by matching sellers policies with buyers intended uses and access criteria.

At the same time, the verification of the validity of Verified Credentials which is a key input in the permissioning process exploits a Self-Sovereign Identity (SSI) agent, that in the current KRAKEN architecture is only a single agent. This means that the corruption of this agent would invalidate the eligibility check as the blockchain would not able to know if the VCs that it receives are valid or not.

Let's consider for example the situation where a data provider publishes a Data Product and a data consumer that wants to buy access to that Data Product doesn't have the required VCs to be eligible for access. If the data consumer is able to corrupt the SSI agent to make it work in a way that it considers his or her VCs valid, the data consumer becomes eligible to buy the Data Product. This situation, provoked by the centralisation of the SSI agent, invalidates the decentralization of the blockchain.

The way data consumers get access to the Data Product, in the batch and distributed analytics case, happens through the SMPC network and the permission to access the data is granted by the permissioning layer (Lynkeus blockchain). This permission is retrieved by the backend from the blockchain network. Once the backend checks that a Data Consumer is eligible for a Data Product, it requests the SMPC network to provide the dataset. The SMPC network receives information from the backend that cannot be proven to come from the blockchain.

Let's consider again the situation where a data provider publishes a Data Product and a data consumer that wants to purchase access to that Data Product doesn't have the right VCs to be eligible to access it. If the data consumer is able to corrupt the backend to make it work in a way that it communicates to the MPC network that he is eligible, the MPC network will behave normally and provide him with the dataset (or more accurately, the analytics results). This situation, provoked by the centralisation of the backend, invalidates the decentralization of the MPC network.

An analogue situation can be seen in the case of payment. The payment is checked on the payment blockchain by the backend, but the Lynkeus blockchain is not able to verify that this information is actually coming from the payment blockchain and needs to trust the backend on its validity.

### 8.1.1 Possible architecture update

A solution to inhibit all these centralisation points is the creation of a new network that would be an extension of the Lynkeus blockchain. A node of this new network would be composed of a node of the Lynkeus blockchain, an SSI agent, a node of the MPC network and a watcher of the payment blockchain.

The connection of an SSI agent to every node of the blockchain gives the capability to the blockchain nodes to be independent in the verification of the validity of the VCs. In this way, even if an attacker is able to corrupt the SSI agent connected to the backend, the nodes of the blockchain, that can independently check the validity of the VCs, would not validate the transaction and the attacker would not succeed.

The same principle applies to the payment blockchain watcher.

In the case of the MPC network, the MPC nodes would still be connected to each other, but in this case, the updates regarding the eligibility of buyers to download certain datasets comes directly from the blockchain (more specifically comes from the trusted node of the blockchain belonging to the same organisation of the MPC node). In this setting, every node of the MPC network is independent of the verification of the validity of the transactions. So even if the attacker is able to corrupt the backend or a node of the MPC network, the other nodes would not validate the new transaction.

The entire schema is summarised in the following image:

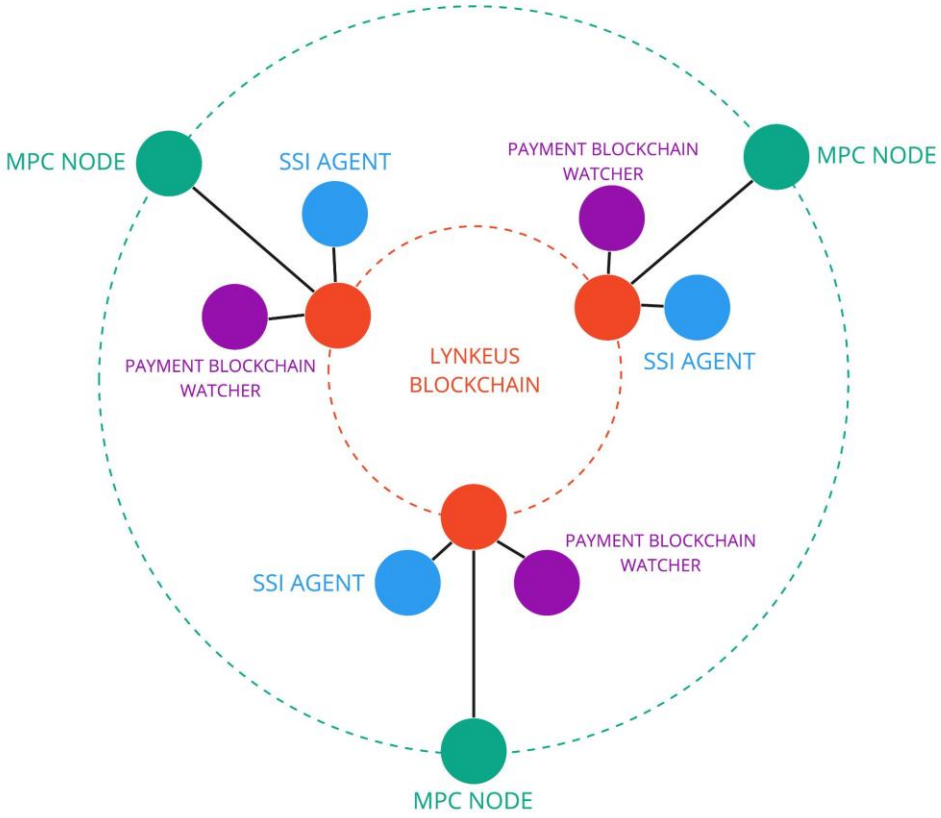


**Figure 18 Fully decentralized design**

This new setting is not centralisation-free. For example the availability of the platform depends still on the backend, but the integrity of the transactions is secured by decentralisation.

Search solution is currently under analysis in terms of their feasibility within the project and actual value in terms of security risk mitigation.

# 9 Conclusion

The Kraken marketplace architecture is in a very advanced stage of definition as main infrastructural components are currently being developed and integrated in view of the first release of the platform scheduled for July 2021 (D5.5).

All key integration aspects have been defined despite high levels of complexity of the underlying use cases, encompassing issues of ethics and regulations, user experience and market requirements, along with very different levels of technological maturity in each of the integrated components. As, for instance, TEX's Streamr network is in public availability, the Aries Framework selected by Atos to implement the SSI framework, it's still in experimental stage in some key areas such as the support of mobile solutions.

As more architectural details are established, especially for the mobile application and the integration between SMPC and the marketplace, the design will turn its focus to remaining issues of centralization at the intersection between SSI and the permissioning layer.

In final version of the architecture (D5.4) we expect to deliver a design that is, on one hand, fully decentralized and on the other able to implement the highest level of privacy and data security, therefore enforcing compliance in all its operations. Such assumption will be tested through a thorough legal analysis in the form of DPIA (data privacy impact assessment) or a similar type of legal and ethical assessment in strict collaboration with WP7.












@KrakenH2020

Kraken H2020

**www.krakenh2020.eu**

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871473