



**BROKERAGE AND MARKET PLATFORM  
FOR PERSONAL DATA**

*D5.6 KRAKEN Marketplace Final Release*

[www.krakenh2020.eu](http://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



## D5.6 KRAKEN Marketplace Final Release

<b>Grant agreement</b>	871473
<b>Work Package Leader</b>	LYNKEUS
<b>Author(s)</b>	Rob Holmes (TEX)
<b>Contributors</b>	Donato Pellegrini (TEX), Giorgi Sheklashvili (TEX), Rosario Meyer (TEX), Davide Porro (ICERT), Sebastian Ramacher (AIT), Davide Zaccagnini (LYNK), Mark Tilen (XLAB), Javier Presa (Atos), Juan Carlos Pérez (Atos)
<b>Reviewer(s)</b>	Davide Zaccagnini (LYNK), Sebastian Ramacher (AIT)
<b>Version</b>	Final
<b>Due Date</b>	31/07/2022
<b>Submission Date</b>	27/07/2022
<b>Dissemination Level</b>	Public

### Copyright

© KRAKEN consortium. This document cannot be copied or reproduced, in whole or in part for any purpose without express attribution to the KRAKEN project.

**Release History**

Version	Date	Description	Released by
v0.1	27/04/2022	Initial draft ToC	Rob Holmes
v0.2	06/05/2022	Updated ToC and inclusion of AIT Privacy Metrics inputs	Rob Holmes
v0.3	21/06/2022	Inclusion of all partner inputs	Rob Holmes
v0.4	29/06/2022	Version for peer review	Rob Holmes
v0.5	11/07/2022	Review by Davide Zaccagnini & Sebastian Ramacher	Davide Zaccagnini, Sebastian Ramacher
v0.6	22/07/2022	Amended following review comments and ready for final quality check	Rob Holmes
v1.0	27/07/2022	Submitted version	Atos

## Table of Contents

1	Introduction .....	12
1.1	Purpose of the document .....	12
1.2	Structure of the document .....	12
2	Marketplace Final Release Overview .....	13
2.1	Technology Components Used in the Final Marketplace Release .....	13
2.2	Major developments provided in first marketplace release .....	16
2.3	Major developments undertaken in final marketplace release .....	16
2.4	Updates to web-based marketplace user flows .....	20
2.5	Final marketplace release architecture .....	20
3	Marketplace Backend API .....	22
3.1	Description .....	22
3.2	Interfaces .....	22
3.3	Deployment.....	24
3.4	Source Code .....	24
3.5	Baseline Technology and Tools .....	25
4	Marketplace Catalogue Database .....	26
4.1	Description .....	26
4.2	Interfaces .....	26
4.3	Deployment.....	32
4.4	Source Code .....	32
4.5	Baseline Technologies and Tools .....	32
5	Marketplace Frontend .....	33
5.1	Description .....	33
5.2	Interfaces .....	40
5.3	Deployment.....	42
5.4	Source Code .....	42
5.5	Baseline Technologies and Tools .....	42
6	Marketplace Smart Contract.....	43
6.1	Description .....	43
6.2	Interfaces .....	43
6.3	Deployment.....	46
6.4	Source Code .....	47
6.5	Baseline Technologies and Tools .....	47
7	Marketplace xDai Watcher .....	48
7.1	Description .....	48
7.2	Interfaces .....	48
7.3	Deployment.....	48

7.4	Source Code .....	48
7.5	Baseline Technologies and Tools .....	48
8	Data Union Joining Server .....	50
8.1	Description .....	50
8.1.1	Development .....	50
8.2	Interfaces .....	50
8.3	Deployment.....	50
8.4	Source Code .....	50
8.5	Baseline Technologies and Tools .....	51
9	Data Union Smart Contract.....	52
9.1	Description .....	52
9.2	Interfaces .....	52
9.3	Deployment.....	52
9.4	Source Code .....	53
9.5	Baseline Technologies and Tools .....	53
10	Streamr Network Integration.....	54
10.1	Description.....	54
10.2	Interfaces .....	54
10.3	Deployment .....	54
10.4	Source Code .....	55
10.5	Baseline Technologies and Tools .....	55
11	Consortium Blockchain Node.....	56
11.1	Description.....	56
11.2	Interfaces .....	56
11.3	Deployment .....	58
11.4	Source Code .....	59
11.5	Baseline Technologies and Tools .....	60
11.5.1	Private Network.....	60
11.5.1	Cache Database .....	60
11.5.1	Smart Contracts .....	60
11.5.1	Application.....	60
11.5.1	Documentation.....	60
12	Marketplace Mobile App .....	61
12.1	Description.....	61
12.2	Interfaces .....	62
12.3	Deployment .....	62
12.4	Baseline Technologies and Tools .....	62
13	MPC Node .....	63
13.1	Description.....	63

13.2	Interfaces .....	63
13.3	Deployment .....	63
13.4	Source Code .....	63
13.5	Baseline Technologies and Tools .....	63
14	Marketplace SSI Agent uSelf Broker .....	65
14.1	Description .....	65
14.2	Interfaces .....	65
14.3	Deployment .....	66
14.4	Source Code .....	66
15	KRAKEN Company Depute Tool .....	67
15.1	Description .....	67
15.2	Interfaces .....	68
15.3	Deployment .....	69
15.4	Source Code .....	69
16	KRAKEN Company Identification Tool .....	70
16.1	Description .....	70
16.2	Interfaces .....	71
16.3	Deployment .....	71
16.4	Source Code .....	71
17	KRAKEN Revocation & Endorsement Registry .....	73
17.1	Description .....	73
17.2	Interfaces .....	73
17.3	Deployment .....	73
17.4	Source Code .....	73
18	Privacy Metrics Tool .....	74
18.1	Description .....	74
18.1.1	Description .....	74
18.1.2	Dataset Independent Inputs .....	75
18.1.3	Privacy Value .....	77
18.2	Interfaces .....	77
18.3	Deployment .....	77
18.4	Source Code .....	77
18.5	Baseline Technology and Tools .....	78
19	Conclusion .....	79
20	References .....	80
21	Annex 1: Selecting Privacy Metrics for KRAKEN .....	81



# List of Tables

Table 1: CA Client API Tasks ..... 58

Table 2: Peer API Tasks..... 58

## List of Figures

Figure 1: Deployment and hosting of integrated technology components in KRAKEN marketplace .....	15
Figure 2: Process flow for creating and publishing the Data Union in the KRAKEN marketplace .....	18
Figure 3: Process flow for members joining the Data Union.....	19
Figure 4: Process flow for purchasing access to a Data Union in the KRAKEN marketplace.....	19
Figure 5: Full Marketplace architecture for KRAKEN Final Release .....	21
Figure 6: Automated code deployment.....	24
Figure 7: Marketplace registration form for data subjects or natural persons representing a company .....	33
Figure 8: Browsing the KRAKEN marketplace data catalogue .....	34
Figure 9: Publishing and setting preferences for access to batch Data Products.....	34
Figure 10: Option to check privacy rating prior to publishing a batch Data Product.....	35
Figure 11: Part of the analytics Data Product publishing workflow.....	36
Figure 12: Computation basket for purchasing analytics Data Products .....	36
Figure 13: Part of the Data Union publication workflow, showing functionality to browse and add data streams to the Data Product.....	37
Figure 14: Option to delete a Data product in user's dashboard .....	38
Figure 15: Personal Dashboard showing a buyer's purchased Data Products .....	38
Figure 16: Dashboard page showing users subscribed to a data provider's Data Products .....	39
Figure 17: A view of the invoice generated when a user opts to pay with fiat for batch Data Products only .....	39
Figure 18: Option to delete a marketplace user's account.....	40
Figure 19: Individual network nodes .....	59
Figure 20: User login to the KRAKEN Marketplace Mobile Application .....	61
Figure 21: Data Products search in the KRAKEN Marketplace Mobile Application .....	61
Figure 22: View of active subscriptions to Data Products in the Marketplace Mobile Application.....	62
Figure 23: Ledger uSelf Broker deployment .....	66
Figure 24: Depute Tool Components .....	68
Figure 25: Depute Tool Components .....	69
Figure 26: KCIT Components .....	70
Figure 27: KCIT web interface.....	71
Figure 28: Dataset independent inputs to compute the privacy metrics .....	75
Figure 29: Dataset dependent inputs for the privacy metrics .....	76
Figure 30: Example result from the privacy metrics computation .....	77



## List of Acronyms

Acronym	Description
API	Application Programming Interface
APK	Android Package Kit
AWS	Amazon Web Services
BSD	Berkeley Software Distribution
CA	Certificate Authority
CSV	Comma-Separated Values
D	Deliverable
DID	Decentralized Identifier
EBSI	European Blockchain Services Infrastructure
ERC-20	Ethereum Request for Comments-20
EU	European Union
GPS	Global Positioning System
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ID	Identifier
JS-NaCl	Java Script-Networking and Cryptography library
JSON	JavaScript Object Notation
KCIT	KRAKEN Company Identification Tool
KRER	KRAKEN Revocation & Endorsement Registry
MSP	Managed Service Provider
MESH	Medical Subject Headings
MHMD	My Health My Data
MPC	Multi Party Computation
QR	Quick Response (Code)
REST	Representational State Transfer
SSI	Self Sovereign Identity
T	Task
TLS	Transport Layer Security
TPS	Transactions Per Second
URL	Uniform Resource Locator
VC	Verifiable Credential
VM	Virtual Machine
WP	Work Package

## Executive Summary

This document describes the software of the final KRAKEN marketplace implementation, covering all modules, programs and tools used for the final marketplace release. It is the final deliverable of Task 5.4 – Backend/Frontend Development within Work Package 5 – Reference platform implementation, pilot’s integration and validation.

The general objective of Work Package 5 is to design and develop a fully functional data marketplace and for the KRAKEN general infrastructure to be tested in the health and education pilots. This final marketplace release consists of an integrated infrastructure that facilitates the exchange of data using three data exchange modalities. The first is the encrypted batch data transfer through directory sharing, the second is privacy-preserving analytics using MPC, and finally the third is data streams managed under the Data Unions framework (See [Section 2.3](#)).

Deliverable 5.6 provides a description of each component used within the final marketplace release, describing the interfaces, deployment guidance, source codes and background technologies and tool usage.

A short description of the components described in this document is provided below:

- Marketplace Backend API ([Section 3](#))
  - A REST API that acts as the central component of the KRAKEN marketplace, it handles all requests that are received or need to be sent to all other components within the marketplace architecture.
- Marketplace Catalogue Database ([Section 4](#))
  - A sub-component of the marketplace backend used to store all associated information or metadata related to Data Products published within the KRAKEN marketplace or a user’s account.
- Marketplace Frontend ([Section 5](#))
  - A component of the marketplace that users directly interact with to perform all of the marketplace operations. Examples include registration, login, Data Products publication, browsing and purchase.
- Marketplace Smart Contract ([Section 6](#))
  - Registers published Data Products on the xDai blockchain, allowing marketplace users to send payments to data providers using an ERC-20 token (DATA).
- Marketplace xDai Watcher ([Section 7](#))
  - Provides the Marketplace Backend API with information about new events occurring on the Marketplace Smart Contract on the xDai blockchain. New events could be Data Product publications, modifications or subscriptions (purchases).
- Data Union Joining Server ([Section 8](#))
  - Handles new Data Union member join requests, adding members’ wallet addresses to the Data Union Smart Contract so that revenues in cryptocurrency can be tracked and withdrawn by Data Union members.
- Data Union Smart Contract ([Section 9](#))
  - A Data Union Data Product is deployed on the Data Union Smart Contract, which facilitates the allocation of shares of payments to members of the Data Union.
- Streamr Network ([Section 10](#))
  - A third-party component for transporting data streams associated with KRAKEN Data Union Data Products over a decentralized scalable real-time messaging network.
- Lynkeus Permissioning Blockchain Node ([Section 11](#))
  - The Lynkeus blockchain is used for data access permissions and policies management and enforcement. It determines data consumer eligibility to access Data Products by performing a check on the stored permissions and policies.
- Marketplace Mobile App ([Section 12](#))

- Provides marketplace users with a mobile environment to allow them to quickly browse Data Products, change permissions and availability of their own Data Products and see how well their Data Products are performing on the data market.
- MPC Node ([Section 13](#))
  - The MPC Network provides a mechanism for key distribution and to perform secure distributed computation / analytics on published analytics Data Products.
- Marketplace SSI Agent Ledger uSelf Broker ([Section 14](#))
  - Facilitates interactions between the marketplace and mobile KRAKEN SSI application for processes related to establishing a DID connection, issuing a credential and presenting a proof.
- KRAKEN Company Depute Tool ([Section 15](#))
  - A tool used by companies to manage natural persons acting on behalf of a company. Allows a company to authorize its employee to perform operations in the KRAKEN marketplace on its behalf.
- KRAKEN Company Identification Tool ([Section 16](#))
  - A tool used by the KRAKEN platform to manage the active list of companies that are currently delegating their employees to operate on behalf of the company they work for.
- The KRAKEN Revocation & Endorsement Registry ([Section 17](#))
  - This component stores information related to the validity of the verifiable credentials (or Verifiable Presentations), such as the status (valid, revoked, suspend).
- Privacy Metrics Tool ([Section 18](#))
  - A tool used by users of the KRAKEN marketplace to quantify the level of privacy that data subjects can expect when sharing their batch Data Products within the marketplace.

# 1 Introduction

## 1.1 Purpose of the document

The aim of this Deliverable 5.6 KRAKEN marketplace Final Release (D5.6) document is to provide an overview of the final software components that together constitute the KRAKEN Marketplace, which is the core development item in Work Package (WP) 5, Task (T) 5.4.

As some of the final software components presented in this document were already presented in the previous submitted deliverable in this series (D5.5 KRAKEN Marketplace Initial Release [\[1\]](#)), some Sections of this report will repeat information that was already reported in D5.5 for completeness. But many of these components have been advanced and extended with additional functionalities since the previous D5.5 submission, and these advancements are also documented within this report.

Some of the components integrated with the core marketplace were developed under the scope of WP3 and WP4 (such as the Self Sovereign Identity (SSI) Agent Uself broker and Multi Party Computation (MPC) node), these are described at a lower level of detail, with references made to their associated deliverable documentation should the reader wish to seek further information. But they have been included in this report because they are tightly integrated, and because they provide crucial functions that are relied upon by the marketplace system.

For each component, a description is provided of its purpose, list of interfaces, deployment, source code and which baseline technologies or tools have been used to build them.

## 1.2 Structure of the document

This document commences with an introduction and high-level overview of the final marketplace and integrated partner components from WP3 and WP4 ([Section 2](#)). Dedicated sections for each of the modules, programs and tools developed within KRAKEN and used within the final Marketplace release then follow. These include the Marketplace Backend Application Programming Interface (API) ([Section 3](#)), Marketplace Catalogue Database ([Section 4](#)), Marketplace Frontend ([Section 5](#)), Marketplace Smart Contract ([Section 6](#)), xDai watcher ([Section 7](#)), and the Data Unions Joining Server ([Section 8](#)) developed by TEX.

A brief description of the Data Union Smart Contract ([Section 9](#)) and Streamr Network ([Section 10](#)) is then also provided. These are external components developed outside of the KRAKEN project within the Streamr Project and leveraged within the final KRAKEN marketplace. The former component is key to allocating cryptocurrency payments between Data Union members, whilst the later provides a decentralized means for transporting data streams in real-time between data providers and data consumers. The Lynkeus Blockchain Node ([Section 11](#)) and Marketplace Mobile App ([Section 12](#)), both developed by Lynkeus, are then described.

From here the document moves on to describe the external components integrated with the marketplace and developed by partners under the scope of the other two KRAKEN pillars of SSI and crypto tools. It describes the MPC node ([Section 13](#)) from XLAB, the SSI agent uSelf Broker ([Section 14](#)) and KRAKEN Revocation & Endorsement Registry ([Section 17](#)) from Atos, the KRAKEN Company Depute Tool ([Section 15](#)) and KRAKEN Company Identification Tool (KCIT) ([Section 16](#)) from Infocert, and finally the Privacy Metrics Tool and API Tool ([Section 18](#)) developed by AIT.

## 2 Marketplace Final Release Overview

This Section provides a brief overview description of the final KRAKEN marketplace release. It first identifies the technology components integrated within the final marketplace release and then moves on to provide:

- A short recap of the major developments provided within the first marketplace release (D5.5);
- An overview of the major developments added to this final release (D5.6);
- An overview of amendments / additions to the web-based marketplace user flows; and
- A diagram of the KRAKEN Marketplace Architecture realised in the final release.

Individual technological components are described in more detail within their respective Sections of this report, and the full scope of functionalities within the KRAKEN marketplace were previously described in D2.7 Design for Marketplace Reference Implementations [\[2\]](#).

### 2.1 Technology Components Used in the Final Marketplace Release

A full list of technological components that together are used in the final KRAKEN marketplace release are listed below. The list also indicates which partners are hosting each individual component within the platform. Responsibility for hosting a component does not mean that the specific partner was also responsible for its development. Some of the components deployed within KRAKEN, which contribute to the functioning of the marketplace platform, are deployed as a single instance, whilst others are deployed as multiple instances hosted by multiple partners. Technological components deployed and hosted by multiple partners include the Consortium Blockchain Nodes hosted by both Lynkeus and TEX, the MPC Nodes hosted by three partners, XLAB, TEX and Atos.

- Hosted by TEX
  - Marketplace Smart Contract
  - Marketplace Backend API
  - Marketplace xDai watcher
  - Marketplace frontend
  - Marketplace Catalogue Database
  - Consortium Blockchain Node
  - Data Unions Joining Server
  - SSI Agent
  - SSI Agent uSelf Broker
  - MPC Node
  - KRAKEN Revocation Registry
- Hosted by Lynkeus
  - Consortium Blockchain Node
  - Marketplace Mobile App
- Hosted by XLab
  - MPC Node
- Hosted by Atos
  - MPC Node
- Hosted by InfoCert
  - KRAKEN Company Depute Tool

- KRAKEN Company Identification Tool

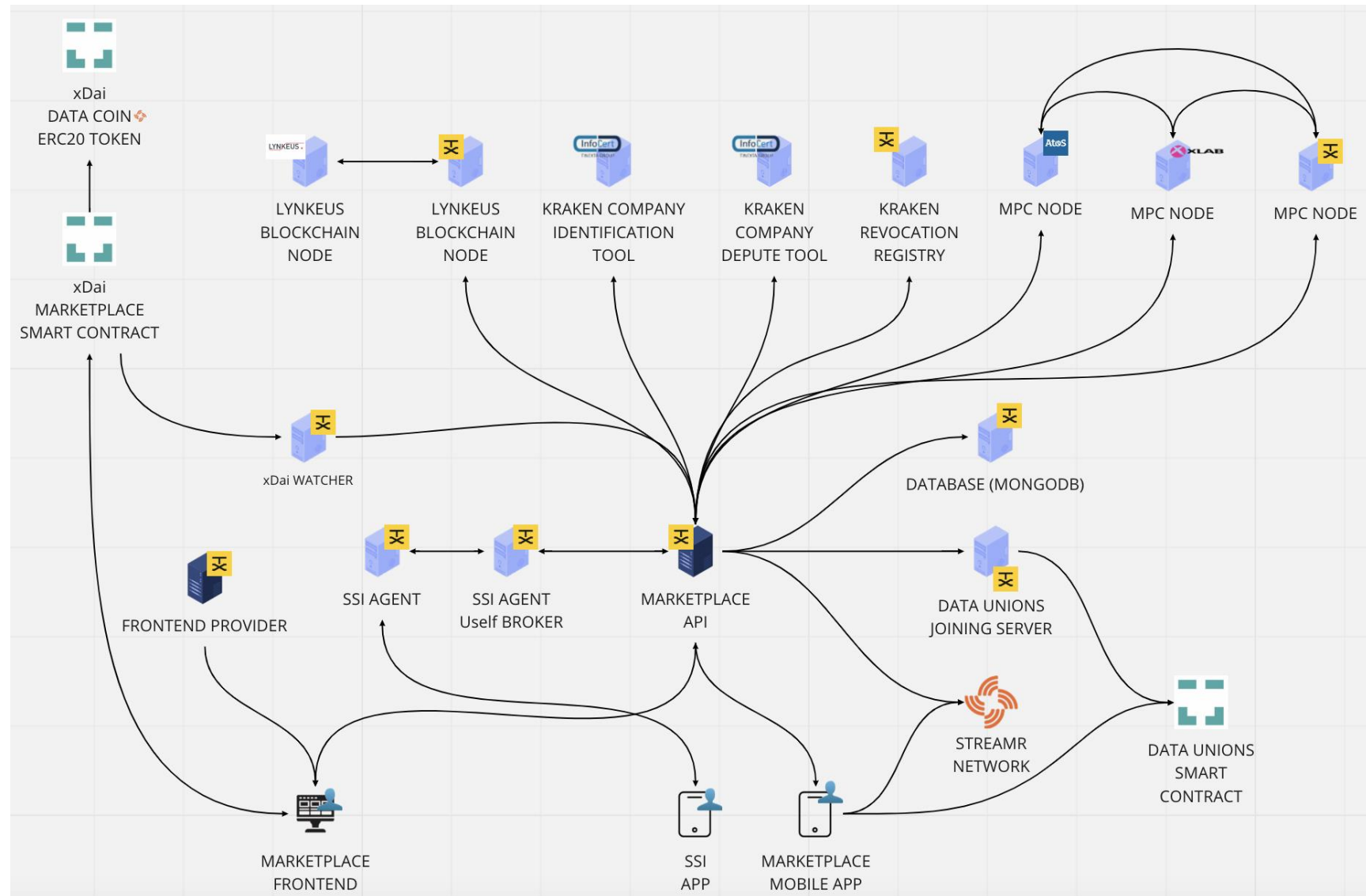
There are two additional third-party components that are integrated and leveraged by the KRAKEN marketplace in the Data Unions use case. These are the Data Unions Smart Contract and the Streamr Network. Both of these two components were developed outside of the KRAKEN project as part of the Streamr Project.

The KRAKEN marketplace leverages a Data Union Smart Contract ([Section 9](#)), which is deployed on the xDai blockchain. A Data Union is a type of Data Product which consists of multiple persons or entities opting in via a third-party application (e.g. mobile app) to share and monetise their data in a Data Product that is managed and administered by a Data Union Manager. The concept of Data Unions was first described in D2.6 Marketplace Technical Specification [\[3\]](#) Section 2.4.2 and also in D2.7 Sections 2.2.2 and 5.1.2.1

The Marketplace Smart Contract ([Section 6](#)) redirects payments to the Data Union Smart Contract which offers a revenue share of funds assigned to members of the Data Union that accumulate in the Data Union contract.

Because the data types to be transacted in the Data Unions data exchange modality implemented within the KRAKEN project are data streams, the marketplace also leverages the Streamr Network ([Section 10](#)), a decentralized Peer-to-Peer pub-sub network that provides end-to-end encrypted data transportation between data providers and data consumers.

The deployment and hosting of the integrated technology components described above are also shown visually in [Figure 1](#) below.



## 2.2 Major developments provided in first marketplace release

To recap, the major developments included within the first marketplace release were as follows:

- Workflow for a user that is not affiliated with an institution or company to register and login to the KRAKEN marketplace.
- Integration with SSI for establishing and storing a users' marketplace credentials in their SSI wallet and presenting them to the marketplace in future logins.
- Workflow for a user to publish a batch Data Product for direct download by a data consumer, including setting of consent preferences for data buyer access.
- Integration with MPC network to provide efficient and decentralized key distribution service between data providers and data consumers in the batch data exchange modality mentioned in the above bullet point.
- Ability for a user to connect a Metamask wallet to make and receive cryptocurrency payments in the KRAKEN marketplace.
- Marketplace home page allowing users to browse Data Products published in the KRAKEN marketplace.
- Workflow for a user to purchase access to batch Data Products in the KRAKEN marketplace.
- Restricted access to Data Products and automated "matching" of eligible users facilitated by the integration of the marketplace with the Lynkeus Blockchain.
- Workflow to edit a previously published batch Data Product.
- A basic dashboard listing a data providers' published batch Data Products in the KRAKEN marketplace.
- A basic dashboard listing batch Data Products that a consumer has subscribed to in the KRAKEN marketplace.

## 2.3 Major developments undertaken in final marketplace release

The following major development items have been added within this final marketplace release.

### Institutional user registration and login

In addition to the registration and login of natural persons or data subjects, this final marketplace release also supports the registration and login of natural persons that represent a company or institution. This is enabled by the marketplace integration with the KRAKEN Company Identification Tool ([Section 16](#)), along with the marketplace integration with the Ledger uSelf Broker of the SSI Agent ([Section 14](#)) and present proof protocol, which allows a user to present a proof of a credential demonstrating their institutional affiliation. The marketplace also checks if a user's institutional or company affiliation is still active or has been revoked by verifying Attorney (company / institutional) Verifiable Credentials (VC) are still active in the Depute Tool ([Section 15](#)). The process for registering in the marketplace as a user representing a company or institution is described in Section 3.4.3.1 of D2.3 - Final KRAKEN Architecture [\[4\]](#), whilst the process for a user to obtain an Attorney VC is described in Section 3.4.3 of D2.3.

### Publication and purchase of privacy-preserving analytical results

The previous marketplace integration with the MPC Network has been extended to allow the publication and purchase of what the KRAKEN team calls, "Analytics Data Products". In addition to publishing a Data Product for batch data download, a marketplace user can now publish a Data Product that is only available for privacy-preserving analytics. In this use case, the data provider is assured of a higher degree of privacy when sharing their data on the KRAKEN marketplace.



The marketplace integration with the MPC network allows the evaluation of basic statistical functions (analytics) on encrypted data without exposing the data itself to the computation nodes or the data buyer, therefore preserving the privacy of the data producers and data sellers involved in the computation. The process flow for how this data sharing modality works has been provided in Section 2.4.2 of D2.7.

### **Publication, purchase and joining of a Data Union**

Through the integration of the Streamr Data Unions Framework and Streamr Network with the KRAKEN marketplace, the KRAKEN team has developed a pilot Data Union demonstrator for the health pilot. In this Data Union demonstrator, individual data subjects can opt-in to offer data from locally stored health apps data in an aggregated Data Product on the KRAKEN marketplace called a Data Union.

A Data Union is managed by a Data Union Operator or Manager. For the purposes of the KRAKEN project's health pilot demonstrator, Lynkeus has taken on the responsibility for this role. Data Union operators have the power to add and remove members. They are responsible for maintaining their Data Unions, including ensuring good data quality and removing members that are not contributing data as they're expected to. Operators are incentivized to perform this work by the Admin fee parameter, a fraction of the incoming Data Union revenue. Typically Operators are creators of both the mobile app and Data Union.

The KRAKEN marketplace mobile application (see [Section 12](#)) developed by Lynkeus, provides individual users with a simple way to opt-in to join the Data Union and publish their mobile health data on a Data Union stream that is delivered to eligible buyers via the Streamr Network. Eligible buyers purchase access to the Data Union data in the KRAKEN marketplace using cryptocurrency, and revenues generated in cryptocurrency are shared between all members that have opted into the Data Union, with a set percentage reserved for the Data Union Operator / Manager.

The following Figures show the flows for creating and publishing the pilot Data Union ([Figure 2](#)), joining individual users/members to the Data Union ([Figure 3](#)), and purchasing a Data Union ([Figure 4](#)).

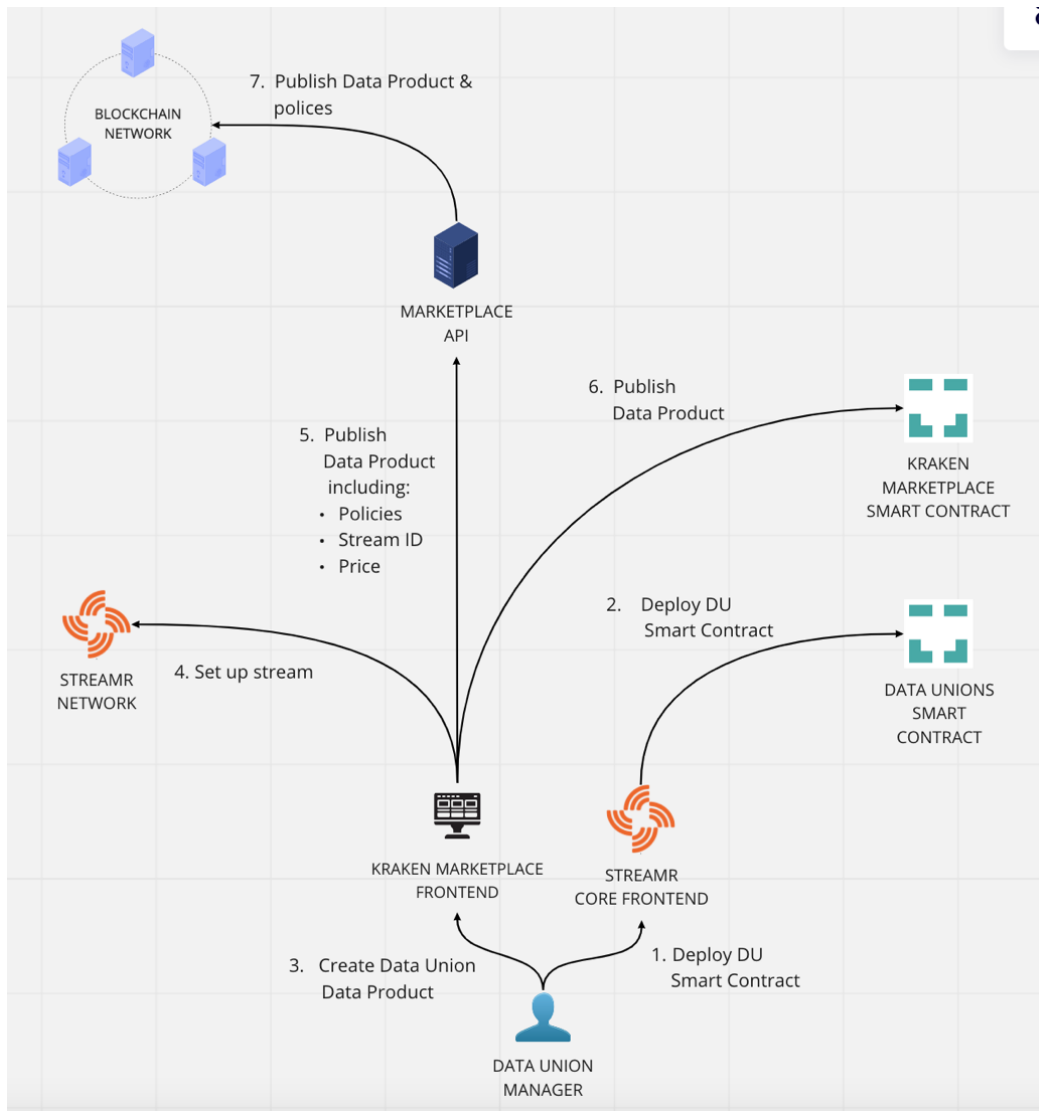
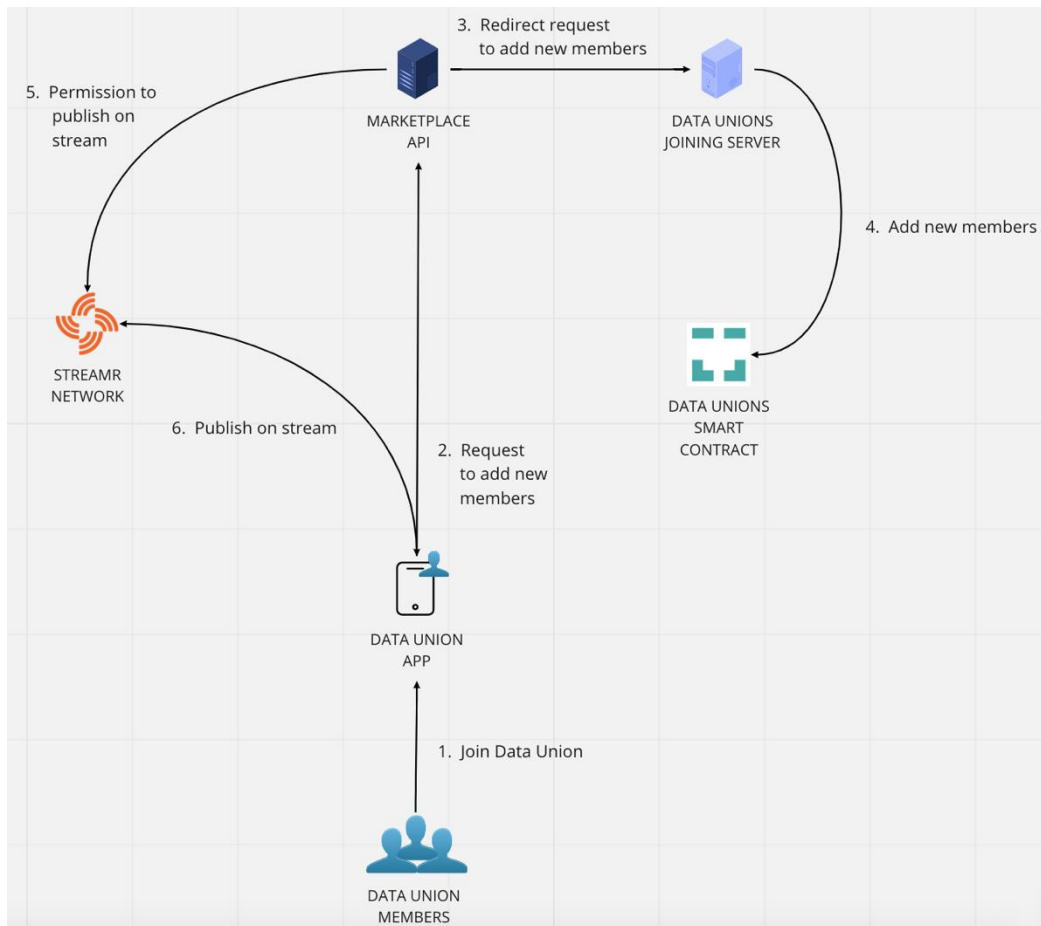
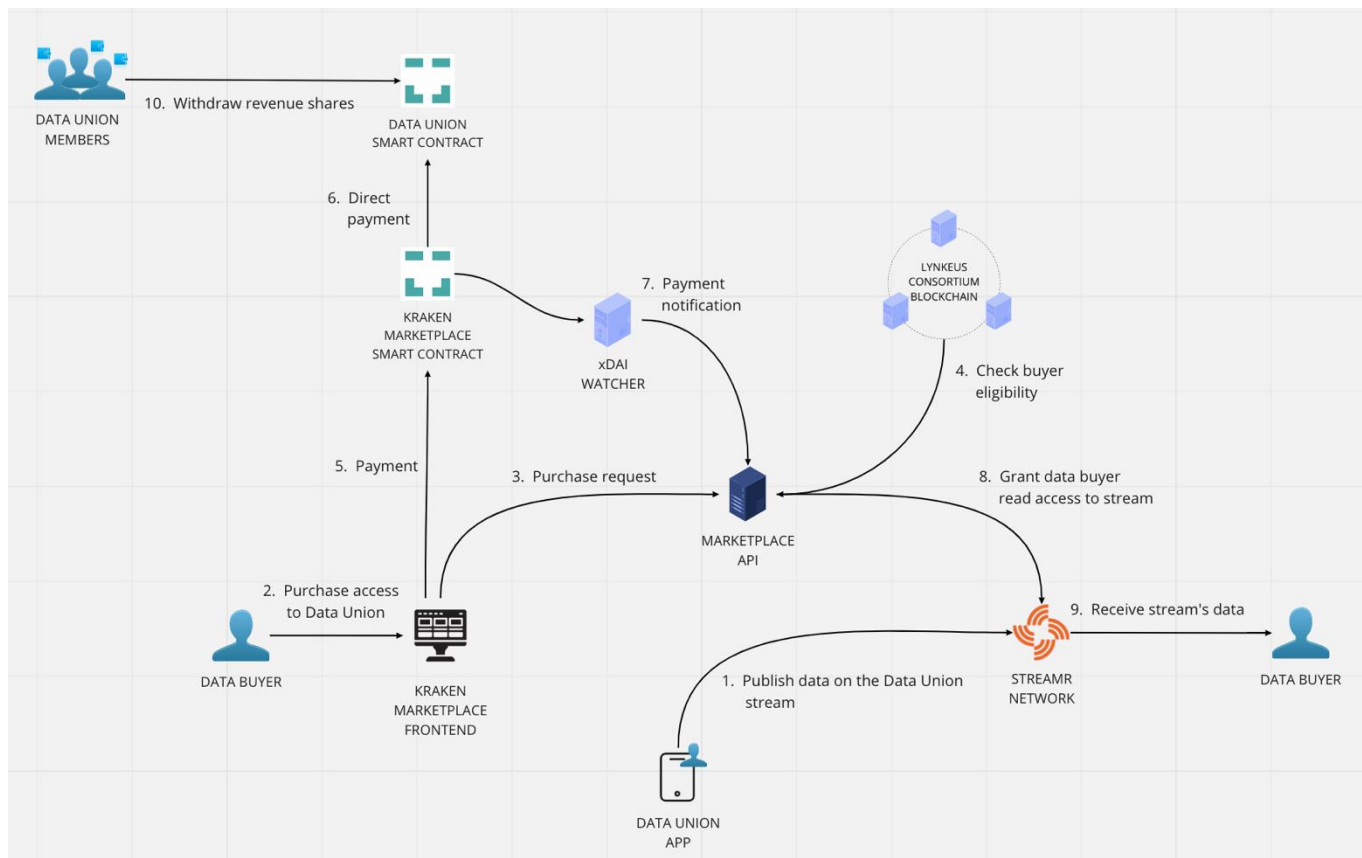


Figure 2: Process flow for creating and publishing the Data Union in the KRAKEN marketplace



**Figure 3: Process flow for members joining the Data Union**



**Figure 4: Process flow for purchasing access to a Data Union in the KRAKEN marketplace**

### Computation of a data subject's privacy metrics

The integration of AIT's privacy metrics tool ([Section 18](#)) into the marketplace frontend allows an individual data subject to measure their privacy prior to publishing a Data Product in the batch data download sharing modality. After the data subject completes a questionnaire, the privacy metrics tool provides them with a score indicating how well they are protected and the level of privacy they can expect after publishing the Data Product.

### Payment with fiat currencies

As described in Section 2.8.2 of D2.5, a demo of an invoicing payments flow has been included in the marketplace for the batch data transfer modality only, and is available only to users representing a company or institution in the marketplace. The demo workflow generates an invoice in Euros, a user can then make a payment using traditional banking transfers outside of the KRAKEN marketplace environment. Once an institutional data provider confirms they have received the payment in their bank account, they can whitelist a buyer in the marketplace to give them access to the Data Product.

### Release of KRAKEN marketplace mobile app

The KRAKEN marketplace mobile application has been integrated with the KRAKEN Marketplace Backend to allow users to use their mobile device to quickly browse Data Products, change permissions and availability of their own Data Products and see how well their Data Products are performing on the market.

## 2.4 Updates to web-based marketplace user flows

In addition to the above major functionality additions, various updates have been made to the web-based marketplace user workflows based on the feedback gathered from the final legal review of the marketplace performed by KUL. This includes:

- Updates to the Data Product publication workflow for batch data, allowing a data provider that publishes special categories of personal data, such as health data, to pre-approve and provide explicit consent for which companies can access their data.
- Updates to the Data Product purchase workflow, to include the collection of additional information from the data buyer regarding their use of the data such as the estimated processing period, the factors that determine this period, and to which country and region the data will be transferred.
- Inclusion of a dashboard page for the data provider, to view a list of data consumers currently subscribed to their data and the relevant information about its use (as described in the previous bullet point) and the data consumers contact details.
- Updates to the Data Product publication page to give a clearer split in the visible questions depending upon if a user is a data subject or a natural person acting on behalf of a company or institution. This provides greater clarity and transparency for the different types of users.
- Addition of the ability to delete Data Products from the marketplace catalogue and delete a user's account.

## 2.5 Final marketplace release architecture

After the integration of the major developments outlined in Sections [2.2](#) and [2.3](#), [Figure 5](#) below shows the full architecture of the marketplace for this final marketplace release, including the interfaces with the other KRAKEN pillars. It also describes the major level interactions between each component.

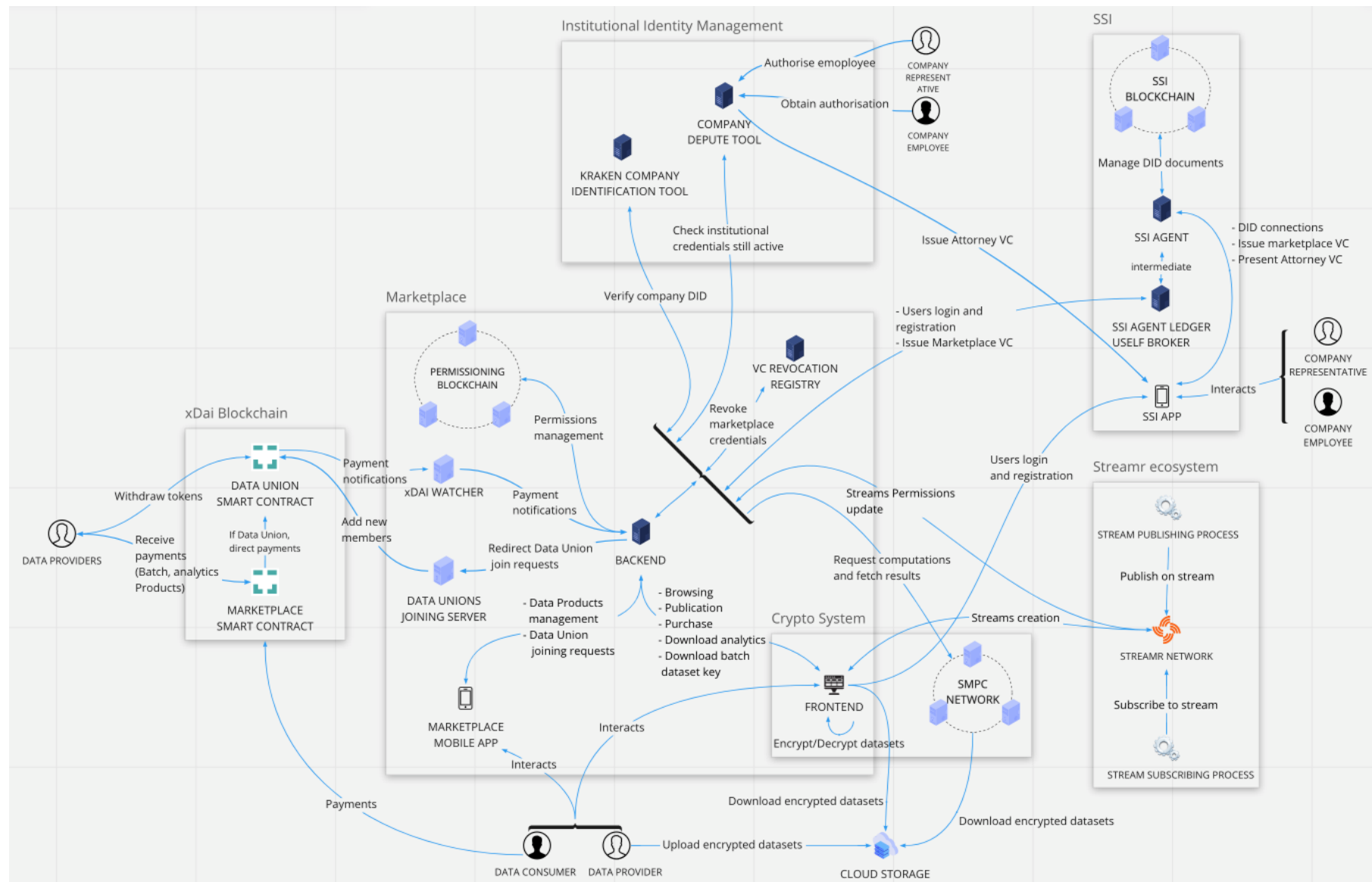


Figure 5: Full Marketplace architecture for KRAKEN Final Release

## 3 Marketplace Backend API

This Section describes the Marketplace Backend API component. This component is fully described in D2.7 Section 4.1.1.1. It is used by the marketplace system to receive any type of client request related to the publication of Data Products and the purchase of access to Data Products.

### 3.1 Description

The Marketplace Backend API is a Representational state transfer (REST) API that handles any request sent by the Marketplace Frontend, the Marketplace xDai Watcher, the KRAKEN SSI app, the Ledger uSelf Broker of the SSI Agent, and the KRAKEN Marketplace Mobile App. These requests enable a large number of operations, with the major level operations identified below:

- Data Product catalogue download, browsing and filtering;
- Data Product publication and modification;
- Data Product publication on xDai;
- Data Product purchase;
- Buyer eligibility checks;
- Data Union member join requests;
- Buyer payment;
- Processing of MPC dataset key request;
- Processing of computation on analytics Data Products request; and
- Data Product browsing and Data Product permissions viewing and editing on KRAKEN marketplace mobile app.

This component also sends requests to the MPC node, the Ledger uSelf Broker of the SSI Agent, the KRAKEN Company Identification Tool, the revocation registry, the Marketplace Catalogue Database, the Consortium Blockchain node, the Streamr Network and the Data Union Joining Server. These requests enable a large number of operations, with the major level operations identified below:

- User registration and authentication for individual marketplace users and institutional users;
- Data Product metadata storage;
- User account data metadata storage;
- Data Product publication on the Consortium Blockchain;
- Buyer eligibility check,
- Dataset key shares storage;
- Permission for a Data Union member to publish on a stream;
- Grant buyer access to a Data Union stream.

### 3.2 Interfaces

A list of interfaces for the Marketplace API are provided below:

- GET/did-connection  
Download did-connection invitation information to perform a Decentralized Identifier (DID) connection with the marketplace SSI agent.
- GET/products

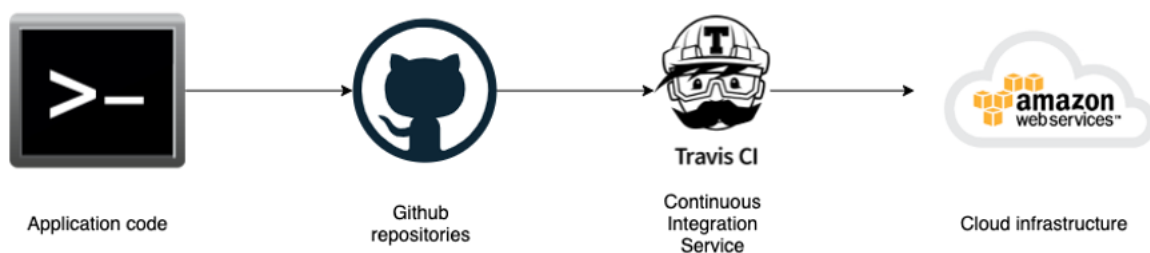
Download the list of public Data Products available on the marketplace.

- POST/products  
Upload a new Data Product on the marketplace.
- GET/products/:id  
Download metadata of a Data Product with a specific identifier (ID).
- PUT/products/:id  
Modify metadata of a Data Product with a specific ID.
- GET/products/:id/streams  
Get list of Data Streams of a Data Product with a specific ID.
- POST/products/:id/keyRequest  
Request the key computation for a Data Product with a specific ID.
- POST/products/:id/deployFree  
Publish a Data Product with a specific ID on the marketplace for free.
- POST/products/:id/setDeploying  
Inform the marketplace that a Data Product with a specific ID is being published on the payment blockchain.
- POST/products/:id/setDeployed  
Inform the marketplace that a Data Product with a specific ID has been published on the payment blockchain.
- POST/products/:id/stateEligibleBuyer  
Request Consortium blockchain eligibility to purchase access to a Data Product with a specific ID for the user sending the request.
- GET/subscriptions  
Download list of subscriptions of the user sending the request.
- POST/subscriptions  
Add a new subscription to a specific Data Product for the user sending the request.
- GET/products/:id/permissions/me  
Download list of permissions of the user that is sending the request.
- GET /users/me  
Download user profile information after login: name, surname and last login
- GET/users/me/products  
Download list of data products owned by the user that is sending the request.
- GET/split.wasm  
Download the web assembly code needed to perform the cryptographic operations for the processing of encryption keys and analytics datasets on the user frontend.
- POST/suggestions  
Download list of suggestions on a provided text for Data Products search functionality.
- POST /dataunion/:dataUnionAddress/joinrequest  
Join request to Data union
- POST /signup  
Signup request
- POST /ssi-webhook

Webhook interface for SSI agent broker

- POST /product/computation  
Request computation for multiple analytics products
- GET /users/:id/productSeller  
Get user information submitted at registration by user ID
- GET/reencryption.wasm.gz  
Request wasm file needed for Batch data product key encryption
- POST /request-proof  
Requesting proof of institutional affiliation during registration (if the user selects institutional registration)
- POST /deleteProduct/:id  
Deleting a Data Product from the marketplace catalogue
- POST /deleteUser  
Deleting a user's account and associated account information

### 3.3 Deployment



**Figure 6: Automated code deployment**

Automated code deployment tools are utilized in the deployment of the project code base. As new versions of the component code are committed to a preconfigured deployment branch on GitHub they are compiled on Travis CI and upon successful compilation they are deployed to a cloud infrastructure hosted by Amazon Web Services.

The code compilation, deployment and cloud infrastructures environment settings are managed with a specific configuration file appended to the code base. Capacity provisioning and health monitoring of cloud infrastructure is handled by Amazon Web Services (AWS) Elastic Beanstalk services.

Travis.yml file guides Travis CI on how the code should be compiled and deployed, e.g., which commands need to be executed and if there are any tests or code linting, how they should be run and where should they be deployed given that previous stages of code compilation are successful.

The deployment includes an Elasticsearch instance provided by Elastic Cloud. The Elasticsearch instance is synchronized with the database by a Moonstache instance deployed on AWS.

### 3.4 Source Code

The code is available on a private repository on GitHub at the following link:

<https://github.com/technology-exploration/KRAKEN-backend>



### 3.5 Baseline Technology and Tools

The Marketplace API is a REST API developed using JavaScript. The libraries the code depends on are express, fabric-network, ws for the web socket communication with the MPC network, mongoose for the database communication and the Lynkeus libraries for the communication with the Consortium Blockchain. For Data Products search functionality in the data catalogue, the Marketplace API relies on Elasticsearch and Monstache. Elasticsearch is a tool that provides smart searching functionality and Monstache is the tool used to keep Elasticsearch synchronised with the Marketplace Catalogue Database.

## 4 Marketplace Catalogue Database

This Section describes the Marketplace Catalogue Database. This component of the marketplace system is the storage system described in D2.7 Section 4.1.1.1. It is used by the Marketplace API to persistently store Data Products descriptions in the form of metadata associated with every Data Product and the user's account details.

### 4.1 Description

The Marketplace Catalogue Database is a MongoDB database instance that is used by the Marketplace API to persistently store all information related to:

- A Data Product's associated metadata;
- The information about registered users' account and verifiable credentials; and
- A log of the DID connections established with the users.

The stored metadata serves the purpose of providing users with the ability to browse and filter Data Products on the marketplace catalogue and describe and define specific details about the Data Product when publishing data descriptions.

### 4.2 Interfaces

A list of Data Models is provided below:

#### DATA PRODUCT

- id: String  
Hexadecimal string of 64 characters representing the product ID. This ID represents the same data product on the xDai blockchain, on the backend and on the Lynkeus blockchain. It's generated for the first time on the Lynkeus blockchain.
- ownerID: String  
Registration Verifiable Credential of the owner of the product
- name: string  
User-provided product name
- description: string  
User-provided product description
- shortDescription: string  
User-provided product short description
- university: string  
Data product university name
- studyProgram: string  
Data product study program name
- course: string  
Data product course name

- **Tags: [String]**  
List of tags to categorise a data product belonging to the health pilot. These tags are selected between the ones available in the Medical Subject Headings (MeSH) ontology. The integration with the ontology is specified in D3.3
- **fileStructureAndFormat**  
set of parameters describing the dataset
  - o **filename: String**  
Name of the dataset file
  - o **format: String**  
Format of the dataset file
  - o **filesize: Number**  
Size of the dataset file
- **owner: String**  
Data Product owner name/pseudonym
- **imageUrl: String**  
Uniform Resource Locator (URL) of the Data product image
- **state: String**  
Deployment state of the Data product. Available options: undeployed, deploying, deployed
- **created: Date**  
Creation date of the Data product
- **updated: Date**  
Latest update date of the Data product
- **minimumSubscriptionInSeconds: Number**  
Minimum amount of seconds a subscription can be purchased or extended
- **ownerAddress: Address**  
xDai address authorised to apply changes to the Data Product
- **beneficiaryAddress: Address**  
Destination address for subscription tokens
- **pricePerSecond: NumberString**  
Data Product price per second
- **priceCurrency: ContractCurrency**  
User-selected Data Product price currency. Available options are DATA for crypto payments and EUR for fiat payments
- **timeUnit: TimeUnit**  
Data Product time unit chosen by the user between: hour, day, week, month
- **category: String**  
Product category: Diploma, transcript, study status, course grade

- price: NumberString  
Data Product price per time unit
- isFree: boolean  
Data Product payment requirement binary indicator
- type: ProductType  
Data Product type chosen by the user between: Batch, analytics and real time stream
- sector: string  
Market sector of the Data Product chosen between health and education
- anonymizeDataset: boolean  
Data product anonymisation binary indicator
- requiresWhitelist: boolean  
Binary parameter indicating if a Data Product is provided with a whitelist
- policies  
Access control parameters selected by the user
  - marketing: Boolean,  
Marketing as a purpose of use for published Data Product
  - publicly\_funded\_research: Boolean,  
Publicly funded research as a purpose of use for published Data Product
  - private\_research: Boolean,  
Private research as a purpose of use for published Data Product
  - managment: Boolean,  
Management or improvement of business services as a purpose of use for published Data Product
  - automated: Boolean,  
Automated decision-making, eg. artificial intelligence (including profiling) as a purpose of use for a published Data Product
  - categories: Boolean,  
Select all buyer type categories as permitted to access Data Product
  - publicHospitals: Boolean,  
Public hospital can access Data Product
  - privateHospitals: Boolean,  
Private hospital can access Data Product
  - privateResearch: Boolean,  
Private Research Centres can access Data Product
  - publicResearch: Boolean,  
Public Research Centres can access Data Product
  - other: Boolean,  
Other non-profits can access Data Product
  - governments: Boolean,  
Government bodies can access Data Product
  - privateCompanies: Boolean  
Private companies (including consultancies, services, technology) can access Data Product
- keyShares: [[Number]]  
Shares of the dataset encryption key encrypted for every for the MPC nodes.

- **datasetUrl:** string  
URL for the download of the encrypted dataset
- **streams**  
List of streams IDs
- **purposes:** Purposes  
List of purposes specified by subjects other than the Data Product publisher for sharing their data as stated in their consent.
- **Boolean fields:**
  - **marketing:** Boolean,  
Marketing as a purpose of use for published Data Product
  - **publicly\_funded\_research:** Boolean,  
Publicly funded research as a purpose of use for published Data Product
  - **private\_research:** Boolean,  
Private research as a purpose of use for published Data Product
  - **managment:** Boolean,  
Managment or improvement of business services as a purpose of use for published Data Product
  - **automated:** Boolean,  
Automated decision-making, eg. artificial intelligence (including profiling) as a purpose of use for a published Data Product
  - **study\_recommendations:** Boolean,  
Study recommendations as a purpose of use for published Data Product
  - **job\_offers:** Boolean,  
Job offers as a purpose of use for published Data Product
  - **statistical\_research:** Boolean,  
Statistical research as a purpose of use for published Data Product
- **dataShareCountries**  
Which countries can access the Data Product (European Union (EU), Third countries, all others)
- **columnVariables**  
Column variable names of published Comma-Separated Values (CSV) file for the purpose of an analytics Data Product
- **numberOfRecords**  
Number of records in the CSV file published for the purpose of an analytics Data Product
- **allowedCompanies**  
Allowed companies which can access a batch or Data Union Data Product

## ACCESS ELIGIBILITY

- **userID:** String  
Registration Verifiable Credential of the user requesting access to the Data Product.
- **transactionID:** String  
ID of the blockchain transaction stating the eligibility of the user to buy the Data Product

- **product: ProductSchema**  
Data product for which the user requested access to
- **address**  
Wallet address of the Data Product consumer

## SUBSCRIPTION

- **userID: String**  
Registration Verifiable Credential of the user subscribing to the Data Product
- **address: String**  
Address of the user which subscribed to product
- **endsAt: Date**  
Expiry date of the subscription
- **dateCreated: Date**  
Creation date of the subscription
- **lastUpdated: Date**  
Latest extension (if any) of the subscription
- **product: ProductSchema**  
Data Product which the user is subscribed to

## USER ACCOUNT

- **state: Number**  
Deployment state of the User's verifiable credential
- **registrationInfo**  
Verifiable credential content
  - o **ID: String**  
Unique ID of the credential
  - o **firstName: String**  
User's first name
  - o **givenName: String**  
User's surname
  - o **email: String**  
User's email
  - o **countryOfResidence: String**  
Country of residence
- **didConnection**
  - o **state: Number**
  - o **id: String**
  - o **invitationID**
- **sharingRadioButton: String**

Indicates if a user is acting in the marketplace on behalf of an organisation

- **ageRadioButton: String**  
Indicates if a user is more than 18 years old
- **contactName: String**  
Indicates if a user wants to be contacted privately by other users
- **institution: String**  
Name of the institution or company a user works for
- **typeOfInstitution: String**  
Type / category of institution or company a user works for
- **legalSurname: String**  
Legal surname of the representative of the organization
- **legalName: String**  
Legal name of the representative of the organization
- **officerEmail: String**  
Data Protection Officer of the institution or company's email address
- **fiatPayment: String**  
Indicates if a user wants to receive payments in fiat currency for any Data Products they publish in the marketplace
- **invoicingName: String**  
Name of the institution or company to be included on an invoice for fiat payments
- **invoicingAddress: String**  
Address of the institution or company to be included on an invoice for fiat payments
- **invoicingZipCode: String**  
Zip or post code of the institution or company to be included on an invoice for fiat payments
- **invoicingCountry: String**  
Country of the institution or company to be included on an invoice for fiat payments
- **paymentInstructions: String**  
Any payment instructions such as bank account details to be included on an invoice for fiat payments
- **privacyConsent: String**  
User consent to KRAKEN privacy policy
- **provider-consumerConsent: String**  
User consent to KRAKEN data provider – data consumer agreement

### 4.3 Deployment

The MongoDB instance exploited in the KRAKEN marketplace is a cluster deployed and provided by MongoDB's Atlas cloud clusters system.

### 4.4 Source Code

The code including schema definitions and the functions used to update them are included in the Marketplace API repository.

### 4.5 Baseline Technologies and Tools

The database itself is a MongoDB instance. The library used to integrate the database with the Marketplace API is mongoose.



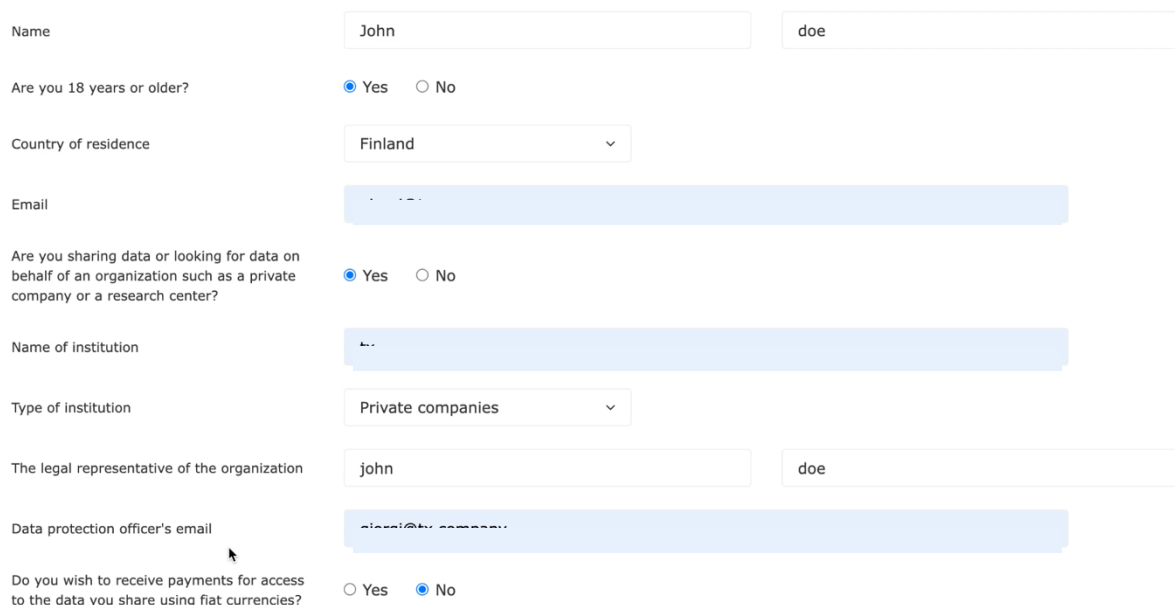
## 5 Marketplace Frontend

This Section describes the marketplace frontend. The marketplace frontend is fully described in D2.7 Section 4.1.1.2. This component of the marketplace system is used to interact with the KRAKEN marketplace and perform all available operations as described in Section 5.1.

### 5.1 Description

The Marketplace Frontend is the software component within the marketplace system that is equipped with a Graphical User Interface (GUI) to allow users of the marketplace to perform the following high-level operations. A select number of Figures (Figures 7-18) are included of the marketplace GUI to help the reader visualise these operations.

- Registration and login on the platform as either a data subject or as a natural person representing a company or institution. If representing a company or institution, the user has to present a proof of its institutional Verifiable Credentials to register on the marketplace (Attorney VC), and a check is also performed that these credentials have not been revoked.



The form is a registration form for data subjects or natural persons representing a company. It contains the following fields and options:

- Name:** Two text input fields. The first field contains "John" and the second field contains "doe".
- Are you 18 years or older?:** Radio button options. "Yes" is selected.
- Country of residence:** A dropdown menu showing "Finland".
- Email:** A text input field containing a masked email address.
- Are you sharing data or looking for data on behalf of an organization such as a private company or a research center?:** Radio button options. "Yes" is selected.
- Name of institution:** A text input field containing a masked name.
- Type of institution:** A dropdown menu showing "Private companies".
- The legal representative of the organization:** Two text input fields. The first field contains "john" and the second field contains "doe".
- Data protection officer's email:** A text input field containing a masked email address.
- Do you wish to receive payments for access to the data you share using fiat currencies?:** Radio button options. "No" is selected.

**Figure 7: Marketplace registration form for data subjects or natural persons representing a company**

- Data Product browsing in the marketplace data catalogue.

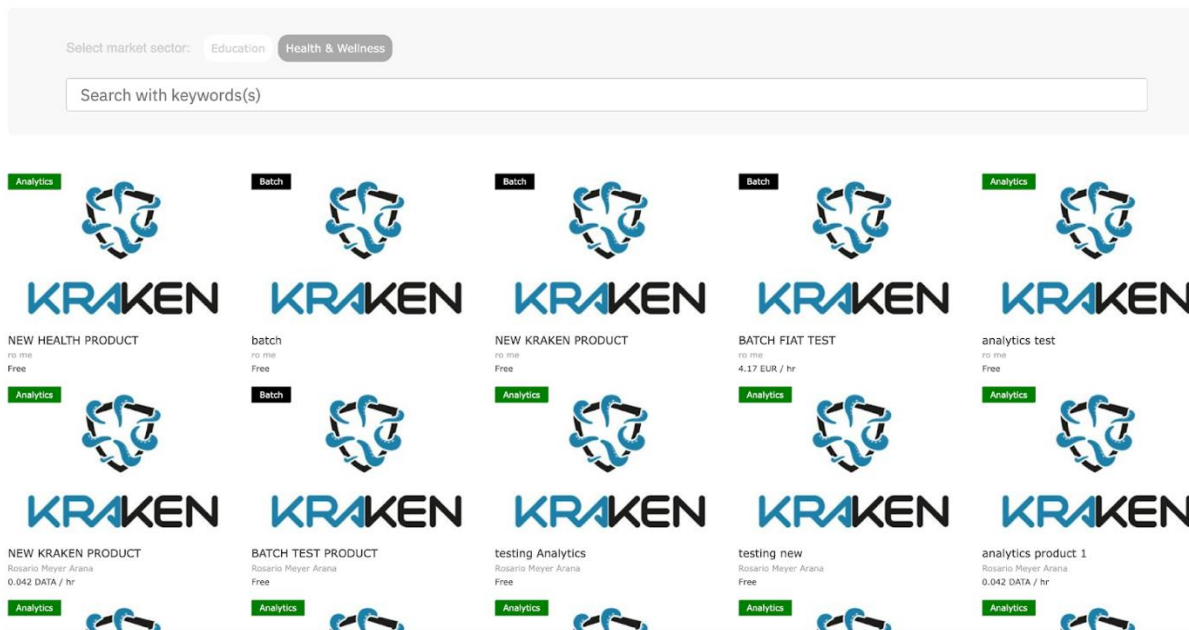



Figure 8: Browsing the KRAKEN marketplace data catalogue

- Batch data publication and setting of preferences for data access, purchase, and download.

Back

Save & Exit
Preview
Continue

Has informed, explicit and free consent been obtained from all data subjects whose data is included to share their data this way?

Be aware that you are the data controller and need to comply with the data protection obligations.

☐ Yes ☐ No

Please list the purposes specified by subjects other than yourself for sharing their data as stated in their consent.

☐ Marketing
 ☐ Publicly funded research
 ☐ Private research

☐ Managment or improvement of business services
 ☐ Automated decision-making, eg. artificial intelligence (including profiling)

Please select which workings and potential significance and envisaged consequences of automated decision making have been approved by the data subjects.

☐ Automated placing of services and product offerings
 ☐ Hiring assessments
 ☐ Clinical risks assessment

☐ Diagnostic or treatment suggestions

Which companies or institutions that are currently registered on KRAKEN marketplace will be able to access and use this data?

Select...

Name

Cover image

Short description

Detailed description

Preferences and legal permissions

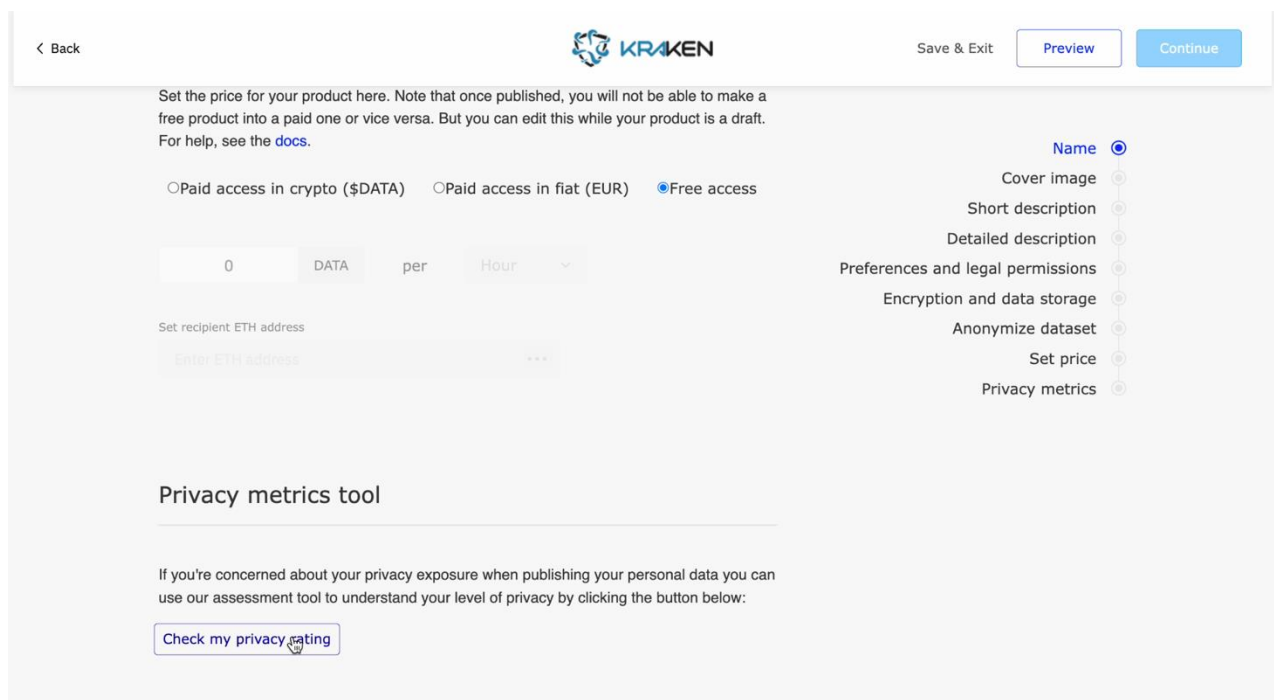
Streams

Shared secrets

Price

Figure 9: Publishing and setting preferences for access to batch Data Products

- When publishing a batch Data Product, as a data subject, a privacy metrics tool is offered to determine the privacy level of the user. Figures showing the graphical interface of the privacy metrics tool are provided in [Section 18](#).



The screenshot shows the KRAKEN Marketplace interface for publishing a batch Data Product. At the top, there is a navigation bar with a "< Back" link, the KRAKEN logo, and "Save & Exit" and "Continue" buttons. Below the navigation bar, a text box instructs the user to set the price for their product, noting that once published, they cannot make a free product into a paid one or vice versa. It also provides a link to "docs" for help.

Below the text box, there are three radio button options for access type: "Paid access in crypto (\$DATA)", "Paid access in fiat (EUR)", and "Free access" (which is selected). Below these options is a price input field with a value of "0", a unit dropdown set to "DATA", a "per" label, and a time unit dropdown set to "Hour".

Below the price input, there is a section for "Set recipient ETH address" with a text input field labeled "Enter ETH address" and a "..." button.

On the right side of the interface, there is a vertical list of steps in the publishing process, each with a circular indicator: "Name" (selected), "Cover image", "Short description", "Detailed description", "Preferences and legal permissions", "Encryption and data storage", "Anonymize dataset", "Set price", and "Privacy metrics".

Below the "Privacy metrics" step, there is a section titled "Privacy metrics tool". It contains a paragraph explaining that if the user is concerned about privacy exposure, they can use an assessment tool to understand their level of privacy by clicking a button below. The button is labeled "Check my privacy rating" and has a mouse cursor hovering over it.

**Figure 10: Option to check privacy rating prior to publishing a batch Data Product**

- Analytics Data Product publication and setting of preferences for data access, purchase through the Computation Basket, and download of the computation result as CSV file.

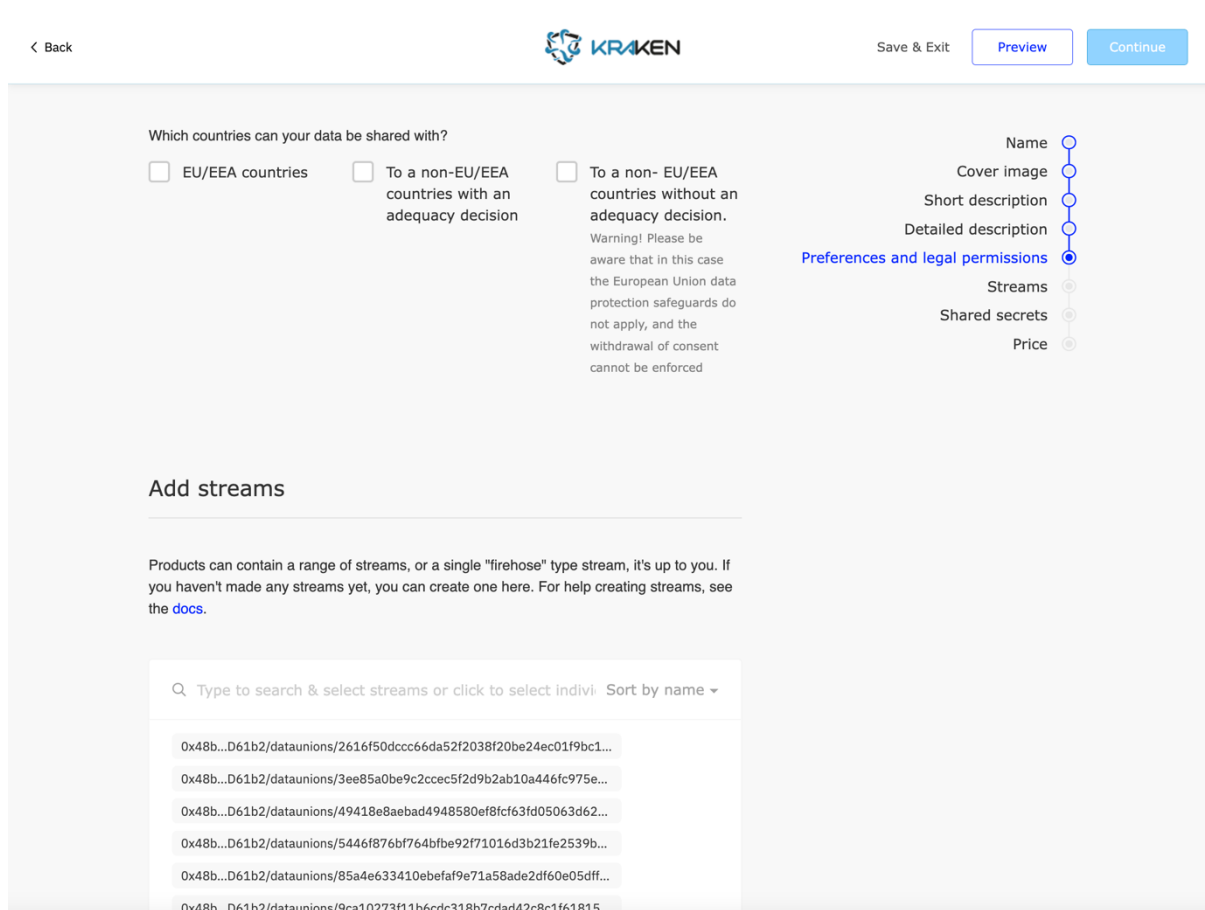
**Figure 11: Part of the analytics Data Product publishing workflow**

You have the following data analytics products waiting for computation

Privacy preserving, remote analysis will be computed based on the following specifications:

**Figure 12: Computation basket for purchasing analytics Data Products**

- In the case of Analytics Data Products, the data is split in the frontend through the MPC network once it is published. When purchased, the data is joined and downloaded by the data buyer.
- Data Union Data Product publication including setting of preferences for data access, adding a user's stream (published on the Streamr Network) to the Data Product. Purchase of access to Data Unions as a stream of data in real-time.



Which countries can your data be shared with?

☐ EU/EEA countries

☐ To a non-EU/EEA countries with an adequacy decision

☐ To a non-EU/EEA countries without an adequacy decision.

Warning! Please be aware that in this case the European Union data protection safeguards do not apply, and the withdrawal of consent cannot be enforced

Name

Cover image

Short description

Detailed description

Preferences and legal permissions

Streams

Shared secrets

Price

### Add streams

Products can contain a range of streams, or a single "firehose" type stream, it's up to you. If you haven't made any streams yet, you can create one here. For help creating streams, see the [docs](#).

Q Type to search & select streams or click to select indivi. Sort by name ▾

- 0x48b...D61b2/dataunions/2616f50dccc66da52f2038f20be24ec01f9bc1...
- 0x48b...D61b2/dataunions/3ee85a0be9c2ccec5f2d9b2ab10a446fc975e...
- 0x48b...D61b2/dataunions/49418e8aebad4948580ef8fc63fd05063d62...
- 0x48b...D61b2/dataunions/5446f876bf764bfbe92f71016d3b21fe2539b...
- 0x48b...D61b2/dataunions/85a4e633410ebefaf9e71a58ade2df60e05dff...
- 0x48b...D61b2/dataunions/9ca10273f11b6cdc318b7cdad42c8c1f61815...

**Figure 13: Part of the Data Union publication workflow, showing functionality to browse and add data streams to the Data Product**

- Delete a user's published Data Product (Applicable to batch, analytics or Data Union Data Products).

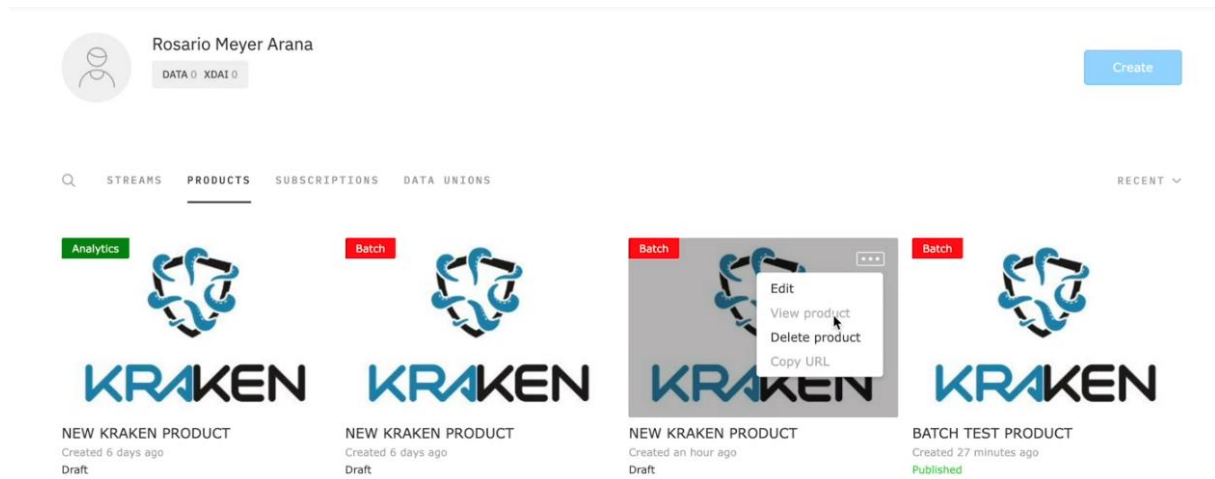


Figure 14: Option to delete a Data product in user's dashboard

- Data buyers can browse their subscriptions in their own personal dashboard.



Figure 15: Personal Dashboard showing a buyer's purchased Data Products

- Data sellers can view users subscribed to their published Data Products on their own personal dashboard.



## Data product subscriptions

### BATCH FIAT TEST

Name	Organisation	Purposes of data processing	Categories of personal data used	Country / region	Transfer based on	Automated decision making	Data stored until
Rosario Meyer Arana	tx	Private research	Health	EU/EEA	Informed consent	No	04/04/2023
<p>Contact details of the data controller: Rosario Meyer Arana - rosario@tx.company</p> <p>Contact details of the Data Protection Officer: Rosario Meyer Arana - rosario@tx.company</p> <p>Safeguards used to protect the data and inform about the risks of the transfer: Rosario Meyer Arana - rosario@tx.company</p>							
ro me	Xyz Shop	Private research	Health	EU/EEA	Informed consent	No	04/04/2023

**Figure 16: Dashboard page showing users subscribed to a data provider's Data Products**

- If the data seller is an institutional user, they can sell batch Data Products in fiat, and can also give access to institutional data buyers to download the batch dataset once the payment is received in their bank account.



DASHBOARDS **MARKETPLACE** rosario@tx.co... ▾

## Invoice

tx  
tx  
Address 1295  
08029, Spain

Invoice to:  
tx  
Rosario Meyer Arana

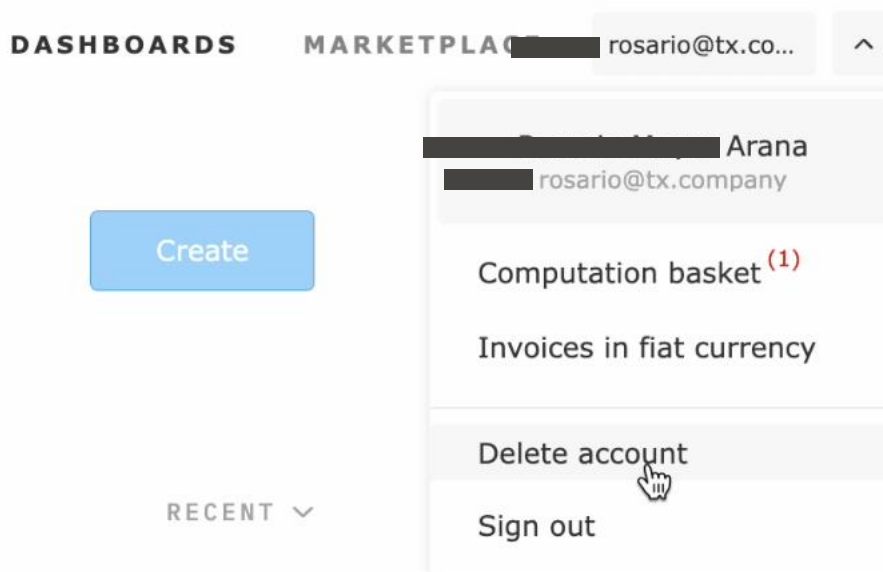
Invoice date: 31 August, 2021  
Terms: Due on receipt  
Due date: 14 September, 2021  
Invoice number: 0001

Data product	Period	Amount
BATCH FIAT TEST	1 year	100 EUR
<b>TOTAL</b>		<b>100 EUR</b> (including VAT 22%)

Payment instructions:  
IBAN ES123456678899600299430 BIC 12345678

**Figure 17: A view of the invoice generated when a user opts to pay with fiat for batch Data Products only**

- Delete a user's marketplace account.



**Figure 18: Option to delete a marketplace user's account**

It should be noted that the marketplace registration and login frontend GUI requires the user to also interact with the Ledger uSelf Mobile App to enable the operations associated with the management of a KRAKEN user's identity. Including establishing a DID connection, the issuing and storage of VCs and presenting a proof of an existing VC. These operations and their associated frontend GUI within the uSelf Mobile App are described in Section 5 of D3.2 Self Sovereign Identity Solution Final Release [5].

The Marketplace Frontend is also integrated with a Metamask Wallet. The purpose of the Metamask Wallet is to allow the marketplace user, depending on if they are acting as a data buyer or data seller, to receive and make payments in cryptocurrency when Data Products are accessed within the KRAKEN marketplace. Marketplace payments for batch and analytics Data Products are facilitated by a solidity Marketplace Smart Contract published on the xDai blockchain. This component is described in [Section 6](#) of this report. Marketplace payments for Data Unions Data Products are also facilitated by the Marketplace Smart Contract, however payments are then directed towards the Data Union Smart Contract, which is described in [Section 9](#).

In facilitating the user operations described in the bullet point list above, the Marketplace Frontend communicates with two components within the KRAKEN marketplace architecture, the Marketplace API and the Marketplace Smart Contract for operations related to the publication, purchase and browsing of Data Products.

Also integrated into the frontend codebase is the code for the Privacy Metrics Tool produced by AIT and described in [Section 18](#).

## 5.2 Interfaces

A list of interfaces for the Marketplace frontend are provided below:

- Marketplace home page  
Browse Data Products that are listed based on the search criteria and the market sector.
- Sign In page



Sign in on the marketplace using the SSI wallet app.

- **Signup page**  
Signup on the marketplace using the SSI wallet app.
- **Connect wallet page**  
Connect the metamask wallet to the platform to perform payment and publication on the marketplace smart contract.
- **Data Product page**  
Show all metadata about a specific Data Product and the policies set by the Data Provider, function to buy the data product and consume it.
- **User products page**  
Browse user's Data Products added to the platform.
- **Edit product page**  
Edit a Data Product's metadata, policies, price, publication.
- **Subscriptions page**  
List user subscriptions.
- **Data Streams page**  
Browse user's Streams added to the platform.
- **Data Union page**  
Browse user's Data Union Products added to the platform.
- **Manage Subscription's page**  
List of the users that are subscribed to the published product. Only accessible to the data owners.
- **Invoices in fiat currency page**  
List of the users that chose fiat payments to purchase the product, access can be managed once the payment is received. Only accessible to the data owners.
- **New product page**  
Choose which type of product to publish.
- **Privacy metrics page**  
For data subjects only, present questionnaire and compute privacy metric to determine the susceptibility of a dataset to revealing private information.
- **Computation basket page**  
for analytics products purchase. When purchase is confirmed, the data is computed and downloaded.
- **Invoice page**  
When confirming the purchase of a batch data product that is paid in fiat, the invoice page is shown for the user to have the payment instructions and the invoice downloaded.
- **Delete account**

Confirm the deletion of a user's account

- Delete product  
Confirm the deletion of a user's product

## 5.3 Deployment

The deployment of this component follows the same process as the one described for the Marketplace API. Please refer to [Section 3.3](#) for further details.

In addition to the configuration files specified in [Section 3.3](#), this repository includes also an .ebextensions file that manages the deployment environment specific settings and configurations, e.g. number of file handlers on Linux operating system.

## 5.4 Source Code

The code is available on a private repository on GitHub at the following link:

<https://github.com/technology-exploration/kraken-frontend>

## 5.5 Baseline Technologies and Tools

The Marketplace frontend is a fork of the Streamr core-frontend repository publicly available on GitHub:

<https://github.com/streamr-dev/core-frontend>

The software is written in JavaScript using the React framework. The relevant libraries already present on the streamr-core-frontend repository and added for the purpose of KRAKEN are web3, Java Script-Networking and Cryptography library (JS-NaCl) and the web assembly library to perform user operations for MPC.

## 6 Marketplace Smart Contract

This Section describes the Marketplace Smart Contract. The Marketplace Smart Contract is described in D2.7 Section 4.1.1.3. This component is used by the marketplace system to perform the publication of a Data Product on the xDai blockchain. It also allows users of the marketplace that are eligible to access a Data Product, based on the data provider's predefined policies, to make payments with an ERC-20 token to purchase access to a data set, computation results of an analytics package or the data of a Data Union - which takes the form of a real time data stream.

### 6.1 Description

The Marketplace Smart Contract is the component responsible for the publication and purchase of Data Products on the xDai blockchain. The Data Product publication and purchase requests sent to this component come only from the [Marketplace Frontend](#). This component is interfaced with the DataCoin ERC-20 smart contract deployed on xDai, allowing users of the marketplace to exploit Streamr's DataCoin token for monetary transactions between data providers and data consumers within the marketplace.

### 6.2 Interfaces

A list of interfaces is provided below:

- `getProduct(bytes32 id)`  
Fetch the info of a product with a specific id.
- `createProduct(bytes32 id, string memory name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)`  
Creates a new product on the smart contract specifying all the parameters.
- `createProductWithWhitelist(bytes32 id, string memory name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)`  
Creates a new product on the smart contract specifying all the parameters and enables it to be whitelisted.
- `deleteProduct(bytes32 productId)`  
Undeploys a product with a specific id.
- `redeployProduct(bytes32 productId)`  
Deploys an existing undeployed product with a specific id.
- `updateProduct(bytes32 productId, string memory name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds, bool redeploy)`  
Updates the product info.
- `offerProductOwnership(bytes32 productId, address newOwnerCandidate)`  
Offers the product ownership to another address.
- `claimProductOwnership(bytes32 productId)`  
Claims the product ownership if the ownership was previously offered by the previous owner.
- `setRequiresWhitelist(bytes32 productId, bool _requiresWhitelist)`  
Sets up a product to require whitelisting or not.

- `whitelistApprove(bytes32 productId, address subscriber)`  
Approve an address to be able to purchase a product.
- `whitelistReject(bytes32 productId, address subscriber)`  
Prevent an address to be able to purchase a product.
- `whitelistRequest(bytes32 productId)`  
Request to be whitelisted for a product with specific id.
- `getWhitelistState(bytes32 productId, address subscriber)`  
Fetch the state of the whitelist on a certain address.
- `getSubscription(bytes32 productId, address subscriber)`  
Fetch the subscription info of an address and a product.
- `getSubscriptionTo(bytes32 productId)`  
Fetch the subscription info of the transaction sender address and a product.
- `hasValidSubscription(bytes32 productId, address subscriber)`  
Checks if the specified address has a valid subscription on the specified product.
- `grantSubscription(bytes32 productId, uint subscriptionSeconds, address recipient)`  
Give free access to a data product to a specific address. This function can only be called by the product owner.
- `buyFor(bytes32 productId, uint subscriptionSeconds, address recipient)`  
Buy the product for a specific address.
- `buy(bytes32 productId, uint subscriptionSeconds)`  
Buy the product for the transaction sender address.
- `updateExchangeRates(uint timestamp, uint dataUsd)`  
Update the exchange rates of the token associated with the smart contract.
- `getPriceInData(uint subscriptionSeconds, uint price, Currency unit)`  
Get the current price of a product in the form of the token associated with the smart contract.
- `halt()`  
Prevent the execution of any function that could provoke modification to the internal state of the smart contract. This function can only be called by the smart contract owner.
- `resume()`  
Revert the actions of "halt()"
- `reinitialize(address datacoinAddress, address currencyUpdateAgentAddress, address prev_marketplace_address)`  
Override the current addresses of the token associated with the smart contract, the currency updater address and the previous marketplace address (if it exists).
- `setTxFee(uint256 newTxFee)`  
Sets a transactions fee that is sent to the smart contract owner on every paid purchase on the smart contract.

- `buyAnalyticsPackage(bytes32[] memory productIDs)`  
Buy an analytics package of products for the transaction sender address. The list of products is provided in the arguments of the function.
- `getPackageByNumOfProducts(uint numOfProducts)`  
Fetch the type of package that the marketplace adopts depending on the number of products.
- `getPackage(string memory id)`  
Fetch the type of package by providing the package id.
- `createPackage(string memory id, uint price, uint paymentToEach, uint8 minProducts, uint8 maxProducts)`  
Create a new package specifying all the parameters.
- `updatePackage(string memory id, uint price, uint paymentToEach, uint8 minProducts, uint8 maxProducts)`  
Updates an existing package specifying all the parameters.
- `deletePackage(string memory id)`  
Deletes an existing package selecting it using the id.
- `packageExists(string memory id)`  
Checks if the package corresponding to the provided id exists.
- `getPackageIDs()`  
Fetches the list of ids of all the packages.

A list of events is provided below:

- `ProductCreated(address indexed owner, bytes32 indexed id, string name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)`  
Notifies about the creation of a new product.
- `ProductUpdated(address indexed owner, bytes32 indexed id, string name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)`  
Notifies about the update of a product providing all the info.
- `ProductDeleted(address indexed owner, bytes32 indexed id, string name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)`  
Notifies about the deletion of a product providing all the info.
- `ProductImported(address indexed owner, bytes32 indexed id, string name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)`  
In the eventuality of the presence of a version 1 smart contract, it notifies about the import of a product providing all the info.
- `ProductRedeployed(address indexed owner, bytes32 indexed id, string name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)`  
Notifies about the redeployment of a product providing all the info.
- `ProductOwnershipOffered(address indexed owner, bytes32 indexed id, address indexed to)`

Notifies about the product's ownership offer to a new owner providing the info about current owner, product id and potential new owner.

- **ProductOwnershipChanged**(address indexed newOwner, bytes32 indexed id, address indexed oldOwner)  
Notifies about the product's ownership change to a new owner providing the info about new owner, product id and old owner.
- **Subscribed** (bytes32 indexed productId, address indexed subscriber, uint endTimestamp)  
Notifies about a new subscription or an extension of an existing subscription providing the product id, the subscriber address and the end of the subscription.
- **NewSubscription**(bytes32 indexed productId, address indexed subscriber, uint endTimestamp)  
Notifies about a new subscription.
- **SubscriptionExtended**(bytes32 indexed productId, address indexed subscriber, uint endTimestamp)  
Notifies about the extension of an existing, not expired subscription.
- **SubscriptionImported**(bytes32 indexed productId, address indexed subscriber, uint endTimestamp)  
In the eventuality of the presence of a version 1 smart contract, it notifies about the import of a subscription providing all the info.
- **ExchangeRatesUpdated**(uint timestamp, uint dataInUsd)  
Notifies about the change of the exchange rates used by the smart contract to convert DATA COIN in dollars or euros and viceversa.
- **WhitelistRequested**(bytes32 indexed productId, address indexed subscriber)  
Notifies about the request of being whitelisted to purchase a product.
- **WhitelistApproved**(bytes32 indexed productId, address indexed subscriber)  
Notifies about the approval of whitelisting request
- **WhitelistRejected**(bytes32 indexed productId, address indexed subscriber)  
Notifies about the rejection of whitelisting request
- **WhitelistEnabled**(bytes32 indexed productId)  
Notifies about the enabling of the whitelisting functionality on a product.
- **WhitelistDisabled**(bytes32 indexed productId)  
Notifies about the disabling of the whitelisting functionality on a product.
- **TxFeeChanged**(uint256 indexed newTxFee)  
Notifies about the change of the marketplace's owner transactions fee.
- **BoughtAnalyticsPackage**(address indexed buyer, string packageID, bytes32[] productIDs)  
Notifies about the purchase of a set of Analytics products using a package.

## 6.3 Deployment

The deployment of the Marketplace Smart Contract is performed on the xDai blockchain. Currently the Smart Contract is deployed and has the following address:

0x4B36680C67EFa16AED5a693726c20dB59428859C

The deployment has been performed using the Truffle library migration functionality. The smart contract can be also monitored on blockscout at the following address:

<https://blockscout.com/xdai/mainnet/address/0x4B36680C67EFa16AED5a693726c20dB59428859C/transactions>

## 6.4 Source Code

The source code is available on a private repository on GitHub at the following link:

<https://github.com/technology-exploration/kraken-marketplace-contracts>

## 6.5 Baseline Technologies and Tools

The Marketplace Smart Contract is a fork of the Streamr-marketplace-contracts repository publicly available on GitHub:

<https://github.com/streamr-dev/marketplace-contracts>

The modifications performed for KRAKEN include the setup of the deployment of the Marketplace Smart Contract to occur on the xDai blockchain instead of Ethereum and other modifications related to the Smart Contract that have contributed also to the Streamr-marketplace-contracts repository. Specifically, the above-mentioned modifications have been applied to make the smart contract independent from an already deployed version 1 of the Streamr marketplace smart contract. The modifications are publicly visible at the following link:

<https://github.com/streamr-dev/marketplace-contracts/pull/44/>

The software is written in Solidity using the React framework. The relevant libraries already present on the Streamr marketplace contracts repository are Openzeppelin, Truffle, web3, mocha.

## 7 Marketplace xDai Watcher

This Section describes the Marketplace xDai Watcher, this component is used by the Marketplace API to get updates on events happening on the Marketplace Smart Contract and perform consequent actions.

### 7.1 Description

The Marketplace xDai Watcher is an intermediary between the Marketplace Smart Contract and the Marketplace API. Its role is to update the Marketplace API about any new events registered on the Marketplace Smart Contract, including Data Product publication, modification, deployment, subscription and deletion.

### 7.2 Interfaces

The Marketplace xDai Watcher does not offer any interface. However, it subscribes to the following list of events on the Marketplace Smart Contract:

- ProductCreated
- ProductRedeployed
- ProductDeleted
- ProductUpdated
- ProductOwnershipChanged
- Subscribed

When any of these events are triggered, the Marketplace xDai Watcher updates the Marketplace API on the following interfaces:

- POST/products/:id/setDeployed
- POST/products/:id/setUndeployed
- POST/products/:id/setPricing
- POST/subscriptions

### 7.3 Deployment

The deployment of this component follows the same process as the one described for the Marketplace API. Please refer to [Section 3.3](#) for further details.

### 7.4 Source Code

The source code for the Marketplace xDai Watcher is available on a private repository on GitHub at the following link:

<https://github.com/technology-exploration/kraken-ethereum-watcher>

### 7.5 Baseline Technologies and Tools

The Marketplace xDai Watcher is a fork of the streamr-marketplace-contracts repository, which is publicly available on GitHub:

<https://github.com/streamr-dev/streamr-ethereum-watcher>



The modifications performed for KRAKEN include the setup of the Watcher to listen to a Marketplace Smart Contract deployed on the xDai blockchain and to send updates on the Marketplace API endpoints.

## 8 Data Union Joining Server

This Section describes the Data Union Joining Server, this component is used by the Marketplace Backend API, which redirects requests from new Data Union members to join the Data Union to the Joining Server. The Data Union Joining Server adds members' wallet addresses to the Data Union Smart Contract so that revenues in cryptocurrency can be tracked and withdrawn by Data Union members.

### 8.1 Description

Data Union Joining server is a software application that provides an interface between Hypertext Transfer Protocol (HTTP) REST and Ethereum Blockchain. This component creates a bridge from HTTP REST to Ethereum Blockchain.

#### 8.1.1 Development

Current Node version is defined in file `.nvmrc` in the project root directory. To use the defined Node version, execute `nvm use` in the project directory. Common development recipes can be found from the Makefile in the project root. For example:

- `make npm-ci`
- `make test`
- `make run`

### 8.2 Interfaces

Interfaces for the Data Union Joining Server are provided below:

- HTTP POST to join Data Union.

Body of the request is in JavaScript Object Notation (JSON) format and contains field `member`. Path of the request contains: `dataUnionAddress`. Both `member` and `dataUnionAddress` are Ethereum addresses. When called with valid parameters this endpoint will join the member to the given Data Union.

### 8.3 Deployment

The application is deployed to Amazon's AWS EC2 environment. Server's IP address is 3.68.33.228 and the server runs on port 8080/tcp.

The application is deployed at path `/home/dataunion/data-union-joining-server`. The application state can be controlled with `systemd` utilities. See `/etc/systemd/system/data-union-server.service` for details.

### 8.4 Source Code

Source code is written in the Node programming language. Source code is stored under version control at GitHub

<https://github.com/technology-exploration/data-union-joining-server>

## 8.5 Baseline Technologies and Tools

Source code repository contains Node 16.15 source code. Makefile for running significant development tasks for the project.

Significant dependencies for data union server are listed below:

- Express 5.x for HTTP server
- Commander.js for parsing command line arguments
- Ethers API 5.x for interacting with blockchain
- Pino API 7.x for logging

Unit testing and development time dependencies are below:

- Mocha for unit testing
- Chai for unit test assertions
- Supertest for testing Express HTTP handler functions

## 9 Data Union Smart Contract

This Section describes the Data Union Smart Contract. The Data Union Smart Contract template was developed as part of the Streamr Project and is leveraged by the KRAKEN marketplace system. The component is used to perform the deployment of a Data Union Data Product and facilitate payment for access to the Data Product, including the allocation of shares of payments to members of the Data Union.

### 9.1 Description

The KRAKEN marketplace has integrated with Streamr's Data Union framework [6], a data crowdsourcing and "crowdselling" solution developed by the Streamr Project. The framework powers applications that enable people to join their valuable data together in a single Data Product and earn by sharing it with interested data consumers.

Data Unions are listed on the KRAKEN marketplace by a Data Union Admin as a type of Data Product for sale. Data producers join and opt-in to share their data to the Data Union using an application. In the case of the KRAKEN health pilot, data producers use the KRAKEN Marketplace Mobile App (Section 12) to opt-in to share data from the health apps on their mobile device. When a buyer purchases access to the Data Product, the revenue in cryptocurrency is automatically distributed to all of the data producers, and the buyer receives access to the aggregated real-time stream of data over the Streamr Network (Section 10).

Data Union Smart Contracts are deployed for individual Data Union Data Products. In other words, each time an individual Data Union Data Product is published, a Data Union Smart Contract is also deployed for that specific Data Union.

The Data Union Smart Contract keeps track of all of the individual Ethereum addresses of the members who have joined the Data Union. It facilitates one-to-many payments for the Data Union each time a single buyer purchases time based-access to a data stream.

The Marketplace Backend API can receive requests for new members to join the Data Union from the Marketplace Mobile App and redirect these along with the Ethereum addresses to the Data Unions Joining Server (Section 8), which allows new members and their addresses to be added to the Data Union Smart Contract.

### 9.2 Interfaces

The interface for this component is a smart contract on the Ethereum blockchain. Smart Contracts that create an individual Smart Contract for a specific Data Union are:

- DataUnionFactory:  
<https://blockscout.com/xdai/mainnet/address/0x82F1b8a9521933ecC41A9062f1eb597D0Ad6e12F>
- DataUnionTemplate:  
<https://blockscout.com/xdai/mainnet/address/0xA7E64C07464DdC69dC1534C14c7c724E5807d42a>

### 9.3 Deployment

Data Unions are fully on-chain, on the xDai blockchain, with a conduit to the Ethereum mainnet. In future it will be possible to run them on any Ethereum-compatible sidechain, however at the moment xDai is the only supported sidechain. The KRAKEN Data Union Smart Contract has been deployed on the xDai blockchain using the Streamr Core app [7]. Streamr Core is a user interface created by the

Streamr Project allowing easy deployment and customising of the parameters of the Data Union Smart Contract, such as the price of the data and the revenue share percentage between its members.

The process for deploying a Data Union Smart Contract is described in full here:

<https://blog.streamr.network/how-to-start-a-data-union/> .

## 9.4 Source Code

The underlying Streamr Data Union smart contracts are open source. The source code can be found in the Streamr GitHub repository: <https://github.com/streamr-dev/data-union>.

## 9.5 Baseline Technologies and Tools

See above for the underlying source code of the Streamr Data Union smart contracts. The smart contract was deployed using the Streamr Core app at the following URL:

<https://streamr.network/core>

Once in Core go to Products > Create > Data Union

## 10 Streamr Network Integration

This Section describes the Streamr Network[8]. The Streamr Network was developed as part of the Streamr Project and was not developed within the KRAKEN project. An integration has been developed between the KRAKEN marketplace and the Streamr Network in order for it to be leveraged in the Data Unions use case of the health pilot, where the decentralized network will be used to transfer data streams produced by the Apple HealthKit Data Union from publishers to subscribers (data buyers).

### 10.1 Description

The Streamr Network is a peer-to-peer network for publishing and subscribing to data in real-time. The KRAKEN marketplace uses the Streamr Network for decentralized data transport for the stream produced by the Apple HealthKit Data Union Data Product in the health pilot.

The Network consists of nodes that interconnect peer-to-peer using the Streamr protocol [9]. Together, the nodes in the Network deliver published streams of messages to all data buyers that subscribe to the Data Union Data Product within the KRAKEN marketplace. The Data Union Operator or Manager uses the Marketplace Frontend to connect the stream produced by the Marketplace Mobile App, and being published on the Streamr Network, to the Data Union Data Product.

The KRAKEN Marketplace Backend API has been integrated with the Streamr Network to grant time-based access to data streams for eligible buyers. Buyer eligibility is determined based on the defined preferences set by the data provider at the time of data publication about who can access and for what purpose. The integration between the Marketplace Backend API and the Lynkeus Consortium Blockchain enables this process of granting access to eligible buyers. For further information on how the Lynkeus Consortium Blockchain determines the eligibility of a data buyer refer to D5.3 Initial KRAKEN marketplace integrated architecture document.

### 10.2 Interfaces

Applications, like the Marketplace Mobile App in the case of the KRAKEN project, publish and subscribe to streams via Streamr nodes. Nodes act as the access points to the Streamr Network. To connect an application to streams it is interfaced with a Streamr node.

Applications can be interfaced with Streamr nodes based on two possible approaches:

- Light nodes: the node is imported to an application as a library and runs locally as part of the application
- Broker nodes: the node runs separately, and the application connects to it remotely using one of the supported protocols

The interfaces for the Streamr Network are described in more detail here:

<https://streamr.network/docs/streamr-network/intro-to-streamr-network>

How to use a Streamr Light node in an application is described here:

<https://streamr.network/docs/streamr-network/using-a-light-node>

How to connect an application to a Streamr Broker node is described here:

<https://streamr.network/docs/streamr-network/connecting-applications>

### 10.3 Deployment

How to deploy and run your own Streamr Broker node is described here:

<https://streamr.network/docs/streamr-network/installing-broker-node>

## 10.4 Source Code

The source code for the Stream Network can be found here:

<https://github.com/streamr-dev/network-monorepo>

## 10.5 Baseline Technologies and Tools

As already indicated in this Section the Streamr Network is an external component not developed within the KRAKEN project, but integrated into the KRAKEN marketplace in order to pilot the Apple HealthKit Data Union in the health pilot.

## 11 Consortium Blockchain Node

This Section describes the Consortium Blockchain Network, a set of organization's hosting nodes that run the smart contracts which handle the marketplace catalogue, user related data and data access control policies.

### 11.1 Description

The Consortium Blockchain Network (built on Hyperledger Fabric) consists of two Organizations: Lynkeus and TEX. Each organization hosts two peer Nodes. Additionally, there is a third logical Organization handling the Ordering Service which is called Orderer Organization and is under the management of Lynkeus and TEX.

The Peer Organizations are joined in a channel in which a Chaincode (Package containing Smart Contracts) is deployed. This Chaincode handles the user preferences and other related data regarding the marketplace, data catalogue operations, as well as enforcing access control policies on the data between a buyer and seller creating agreements.

Marketplace users interact with the Blockchain Network in the following ways:

- They register/enroll to an Organization's Certificate Authority to obtain crypto material (certificate) that allows them to perform chaincode operations.
- They can call chaincode functions via the marketplace UI such as creating the account, creating a product or buying data.

For more information on the Blockchain Network and Hyperledger Fabric refer to Deliverable 5.3.

### 11.2 Interfaces

#### Application (Node JS) API

##### CA Services

- registerAppUser. Register a user on the selected Certificate Authority (CA)
- enrollAppUser. The user enrolls using a Certificate Signing Request and obtains a signed certificate.
- updateUser. Update user data on the CA
- deleteUser. Delete a user from the CA
- isAdmin. Check if a certificate belongs to an admin
- reenrollAppUser. Reenroll a user to obtain a new certificate
- getExpirationDate. Get the max expiration date among all the certificates of a user

##### Cache Queries

- queryUsers. Query all users
- queryUser. Query user by username
- queryProducts. Query all products
- queryProduct. Query product by id
- queryCatalogue. Query the available products (All products with non-expired certificates)
- queryFilteredProducts. Query and filter the catalogue to match the browsing user's preferences



### Block Listening

- `createBlockListener`. Create a listener that fetches each newly appended block
- `removeBlockListener`. Remove a listener
- `handleTransactionData`. Extract all the Events and EventData from the fetched blocks and forward them to the database

### Util

- `connectGateway`. Connect to the Fabric Blockchain Network (all nodes) using a connection profile.

### Signing Offline

- `sendTransaction`. Sign a transaction manually on the user side and send it to the network.

Documentation is generated using jsdoc and will be published at a later stage.

## **Chaincode API**

### User Credentials

- `CreateUser`. Create a user account on the ledger
- `UpdateUser`. Update a user account
- `ReadUser`. Read user data
- `DeleteUser`. Delete a user account

### Data Catalogue

- `CreateProduct`. Create a product
- `UpdateProduct`. Update a product
- `ReadProduct`. Read product data
- `DeleteProduct`. Delete a product
- `BuyProduct`. Calls Agreements Contract to validate the eligibility of the buyer to access this product

### Agreements

- `NewAgreement`: Store a new agreement if the validation is successful and the transaction status
- `UpdateAgreement`: Update an agreement's status

Documentation is generated using godoc and will be published at a later stage.

## **Installation APIs (CLI)**

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 Transport Layer Security (TLS) CA / CA
Enroll user
Re-Enroll user
View list of Identities in CA
Setup and launch CA Server
Setup Organization Managed Service Provider (MSP)

Table 1: CA Client API Tasks

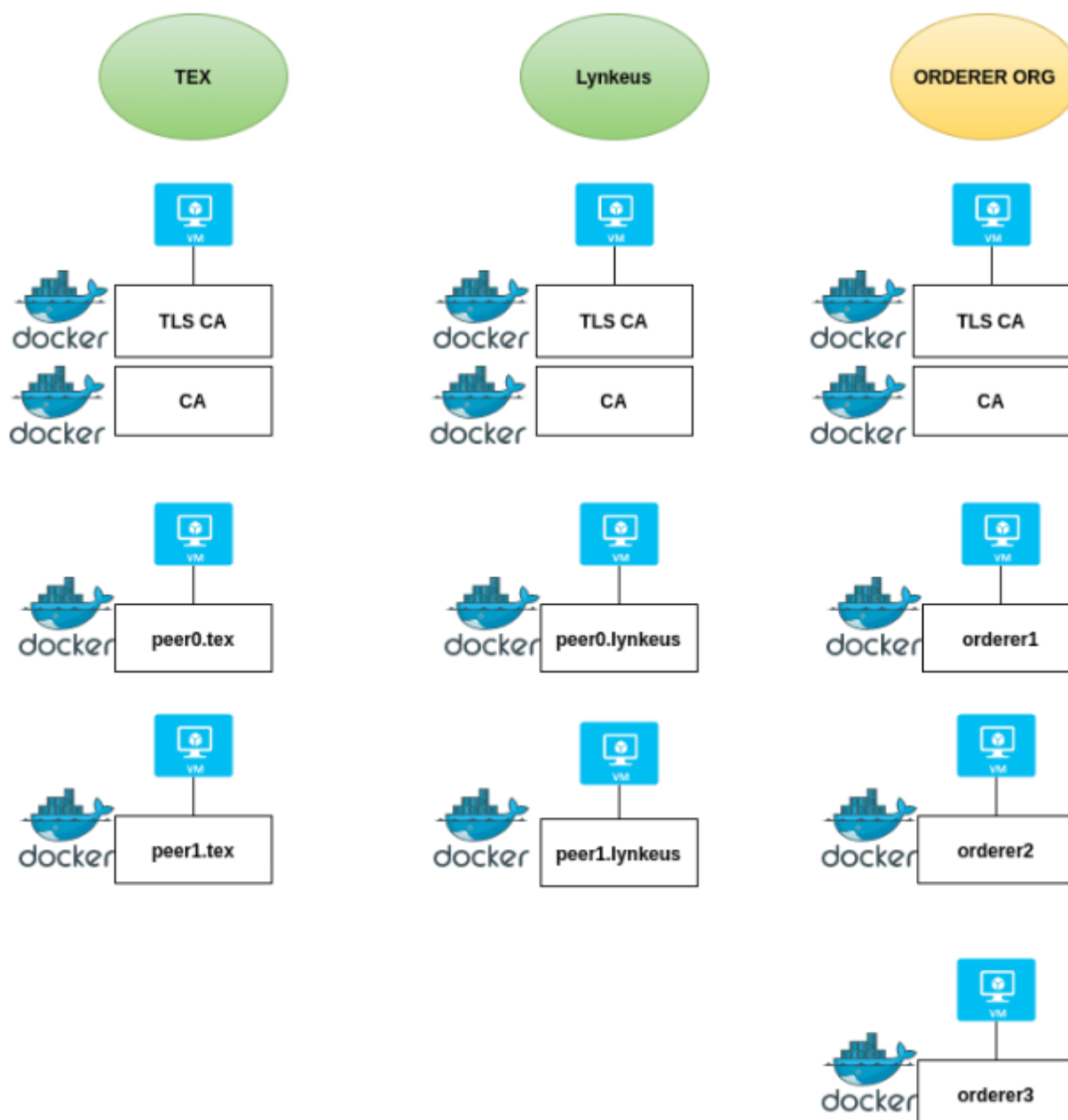
Peer API Tasks	
Creates a channel transaction from profile config	Install chaincode to peer
Submit channel transaction to orderer	Query installed chaincodes on peer
Joins a peer to channel	Approve chaincode as Org
Create and submit anchor peer update transaction	Query approved chaincodes on channel
List 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 2: Peer API Tasks

### 11.3 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 enrolment of nodes, so after the initial enrolment, it will be down if another node will not be joined to the network. Thus, we have used one VM for both CAs for better resource management.

The individual network nodes are depicted in the following Figure.



**Figure 19: Individual network nodes**

The deployment of the network is a collaborative effort between the organizations and requires several steps to be fully set up. The steps are well defined in a file called “DEPLOY.md” in the repository.

## 11.4 Source Code

The Hyperledger Fabric network code is currently stored on a private repository of the developer and is shared with members of other teams.

## 11.5 Baseline Technologies and Tools

### 11.5.1 Private Network

Hyperledger is the framework leveraged in Kraken building on the My Health My Data (MHMD) work on this front, expanded for the specific requirements at hand. Specifically, the Fabric instance can achieve two times more transactions per second (TPS) with an appealing implementation and code structure from a development standpoint.

### 11.5.1 Cache Database

This database is a MongoDB instance.

### 11.5.1 Smart Contracts

HLF smart contracts are written in GO, which is strict in terms of syntax and logic and is thus of great use when writing smart contracts. These contain data access parameters that are computed by the filtering algorithm running on the Blockchain to compute data access permissions.

### 11.5.1 Application

The backend application is written in Node.js and is built mainly using Hyperledger Fabric's Node SDK which utilizes functionalities to provide connection and interaction with the blockchain network. The application is the middleware between the network and the user and has been optimized to achieve high throughput, security, and privacy.

### 11.5.1 Documentation

The smart contracts code is documented using godoc and its standard code documentation practices. In respect of the Node.js application, we used the tool JSDoc.

## 12 Marketplace Mobile App

This Section describes the KRAKEN Marketplace Mobile Application, a mobile version of the KRAKEN marketplace providing certain limited functionalities for marketplace users.

### 12.1 Description

The KRAKEN Marketplace Mobile Application connects the marketplace to its mobile environment to allow users to quickly browse Data Products, change permissions and availability of their own Data Products and see how well their Data Products are performing on the market. In order for the user to connect to the marketplace application the user first needs to scan a Quick Response (QR) code to retrieve a token which will authenticate him/her. Such a QR code is made available to the user after logging in to the browser marketplace application. This enables retrieving information from the marketplace. At that point the user is able to login using his/her credentials (Figure 20).

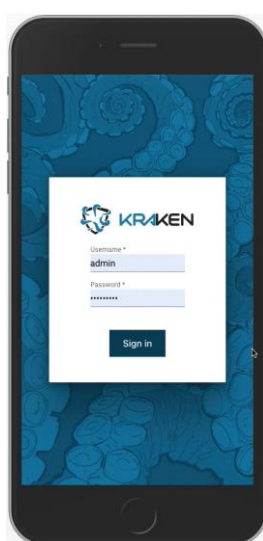


Figure 20: User login to the KRAKEN Marketplace Mobile Application

Data Products can be searched using the same type of semantic search implemented in the web-based version of the KRAKEN data catalogue (Figure 21).

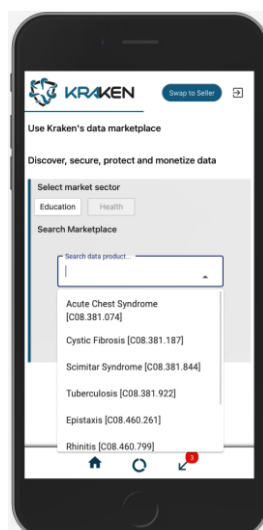


Figure 21: Data Products search in the KRAKEN Marketplace Mobile Application

Active subscriptions to Data Products can be viewed in the app too (Figure 22)



**Figure 22: View of active subscriptions to Data Products in the Marketplace Mobile Application**

Finally, the marketplace application supports offline signing of requests to enable the editing of entries on the blockchain e.g. to change permissions or availability. That is, requests are directly signed on the application and are then sent to the backend. In this way, the expectedly frequent usage of the app, i.e. the management of data access parameters by sellers of personal data, is fully supported even directly on the app, with no additional authentication required.

The application has been in an active state of development and will be finished and tested during the month of July 2022, in preparation for the pilot test in September 2022.

## 12.2 Interfaces

Once the user is connected to the app, information is made available to the user through the backend RESTful API and http/https calls.

## 12.3 Deployment

The KRAKEN Marketplace Mobile app will be downloadable from an Android Package Kit (APK) link.

## 12.4 Baseline Technologies and Tools

The software is written in React native <https://reactnative.dev/>.

## 13 MPC Node

This Section describes the MPC nodes component. This component is used by the KRAKEN platform as a mechanism for key distribution and to perform secure computation / analytics on published datasets. We briefly summarize the component here, however for further details please refer to Deliverable 4.4 Final implementation of cryptographic libraries [10], where the component has been extensively documented.

### 13.1 Description

Secure Multi-Party Computation allows users to evaluate various programs including analytics on encrypted data, without revealing the data itself. This is achieved by splitting the queries among MPC nodes that can jointly and securely perform the desired computation. The privacy of the dataset is guaranteed by the decentralization.

The KRAKEN platform uses MPC as a mechanism for the key distribution and to evaluate various analytics on the datasets, see D2.4 and D2.5 for further details on MPC architecture and protocols and D5.3 Initial KRAKEN Marketplace Integrated Architecture [11] for a detailed explanation of the two applications of MPC.

### 13.2 Interfaces

An MPC node is a service interfacing in the following way:

- Accepts requests from the Marketplace Backend API (Section 3) component.
- Returns results of the computations to Marketplace Backend API (Section 3) component.
- Communicates with other MPC nodes to securely evaluate functions.

All the above communication is done through sockets. See D4.4 for a detailed explanation of the structure of the requests and responses. However please note that while the MPC nodes directly communicate only with the Marketplace Backend API (Section 3), this component serves only as a connector between data sellers, data buyers and the MPC nodes in the KRAKEN platform. No data can be revealed to the Marketplace API (Section 3).

### 13.3 Deployment

MPC nodes can be deployed as Docker containers with specified addresses of the other MPC nodes. In the current release 3 nodes are deployed on the XLAB servers, while in the following releases the nodes will be distributed among the partners: as agreed these are XLAB, ATOS and TEX.

### 13.4 Source Code

All of the source code is available to all the partners in the KRAKEN consortium together with all the other cryptographic tools in a private repository at <https://github.com/krakenh2020/KrakenCryptoTools>. We plan to make this repository public in the future.

### 13.5 Baseline Technologies and Tools

The MPC node component is implemented in Go programming language managing the communication and scheduling of tasks. It uses a fork of SCALE-MAMBA <https://github.com/KULeuven-COSIC/SCALE-MAMBA>, for building and evaluating secure multi-party computations. It uses a homomorphic proxy

encryption protocol for decentralized key management, please see D2.5 KRAKEN Final Technical Design [\[12\]](#) and D5.3 for further information.



## 14 Marketplace SSI Agent uSelf Broker

The SSI Agent deployed by KRAKEN in the marketplace is called Ledger uSelf Broker, being an enterprise-level solution implemented to support the interaction between the user and marketplace and enable communication flow in the processes between them. It is one of the core components of the SSI approach integrated in the architecture of KRAKEN

### 14.1 Description

This component will be located between the Marketplace services and the SSI Server Agent provided by the Hyperledger Aries protocol implementation used in the SSI approach.

It means, this Broker will facilitate the interactions between the different services and processes performed in the marketplace and the consumers of these services using their mobile KRAKEN SSI application.

For this purposes, three processes are implemented in this component:

- Exchange DIDs, this protocol is used to create a connection between the service provider (marketplace) and the user based on DID.
- Issue credential, to enable the issuance of a Verifiable Credentials
- Proof presentation, to initiate the request of the proof.

More information related to the mentioned SSI Server Agent and the other SSI components, i.e., the SSI Mediator Agent or SSI Mobile Agent can be found in the deliverable D3.2 Self-Sovereign Identity Solution Final Release, where it is explained that those agents will use the underlying framework of Hyperledger Aries.

### 14.2 Interfaces

The following interfaces are available entry points of the Ledger uSelf Broker, having several for the usage and some for configuring and managing the server.

The following are dedicated for the usage:

- POST/connections/generate-invitation  
For generating an invitation to be translated into a QR code
- POST/issue-credential/issue  
For issuing a credential
- POST/present-proof/request-proof  
For requesting a proof
- GET/kms  
For managing the cryptographic material. This interface changes from the previous version

It is worth mentioning new functionalities provided by this component related to the management of public DIDs.

- GET/did\_web/register/  
For generating a new DID document (did:web method)

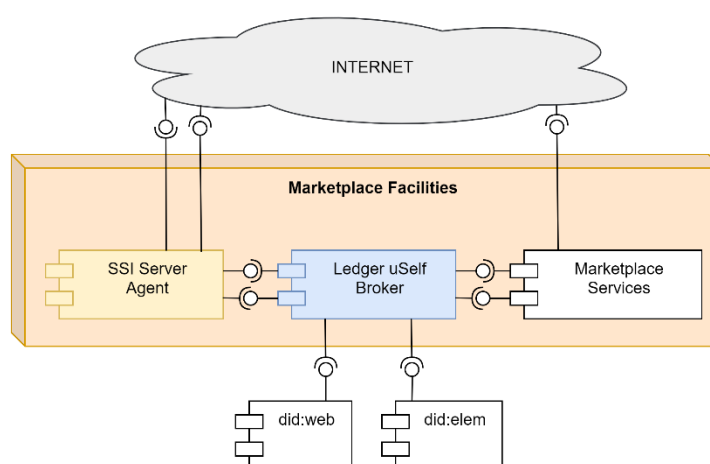
- POST/did-uself/agent/register
- POST/did-uself/register

Both interfaces are for publishing a DID document following a did:elem method

More information regarding the rest of the interfaces can be found in D3.2 KRAKEN project: D3.2 Self-Sovereign Identity Solution Final Release.

### 14.3 Deployment

SSI Ledger uSelf Broker component must be deployed as a docker container following the diagram in Figure 23 and the instructions together with the source code.



**Figure 23: Ledger uSelf Broker deployment**

### 14.4 Source Code

To find the source code for the Ledger uSelf Broker is in a private repository.

There, it is possible to obtain not only the source code but all the needed information for the deployment, configuration and some instructions on the usage.

## 15 KRAKEN Company Depute Tool

The Depute Tool is one of the two software tools developed in the KRAKEN project to manage natural persons acting on behalf of a company.

Each instance of the Depute Tool is used by a company to authorize its employee to perform operations in the marketplace, for example Data Product publication, selling and buying, on behalf of the company.

One instance of the Depute Tool is deployed on the KRAKEN platform for demonstration purposes. To guarantee that only the authorized company's users can operate on the tool, both the Angular 13 front-end and the restful API used by the frontend, are protected. Moreover, the underlying rest go agent's rest API is not exposed.

The Depute Tool's security is delegated to an instance of KeyKloak [13], an open-source identity and access management tool deployed on AWS.

The Depute Tool's user authentication and authorization is based on users and roles configured in KeyKloak, the protocol used is openid-connect [14]. Technically a Depute issued authorization is implemented by an SSI Verifiable Credential.

The tool works using the Aries's Go REST agent implementation; it supports the following Aries protocols:

- DID Exchange
- Issue-Credentials

Issuing and revocation of credentials are based on the underlying framework functionalities (VC Schemas and VC revocation/endorsement). Since revocation/endorsement functionalities are not available in the current Aries-Go implementation [15], KRAKEN will provide a custom implementation.

The tool manages two kinds of SSI entities:

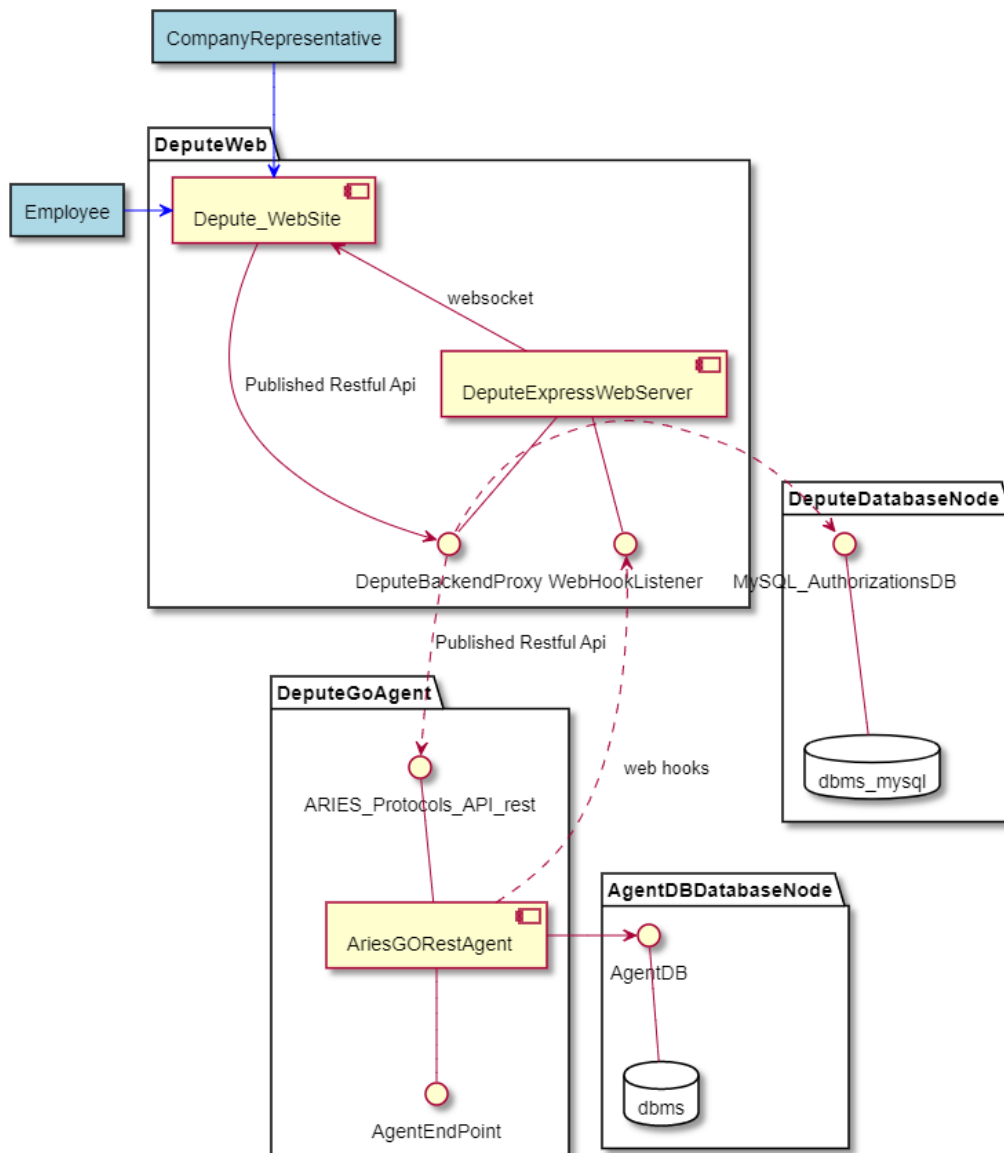
- DID connections
- Verifiable Credentials

### 15.1 Description

The KRAKEN Depute Tool's architecture is organised in the following node/components:

- DeputeWeb Node:
  - DeputeWebSite standard angular 13 web site;
  - DeputeExpressWebServer: an express framework nodejs web server acting as http-proxy;
- DeputeGoAgent Node
  - ArieGOrestAgent: a standard Aries Go REST Agent
- AgentDBDatabaseNode:
  - Apache CouchDB instance.
- DeputeDatabaseNode:
  - MySql db instance.

Figure 24 below shows the Depute nodes/components described in this Section.

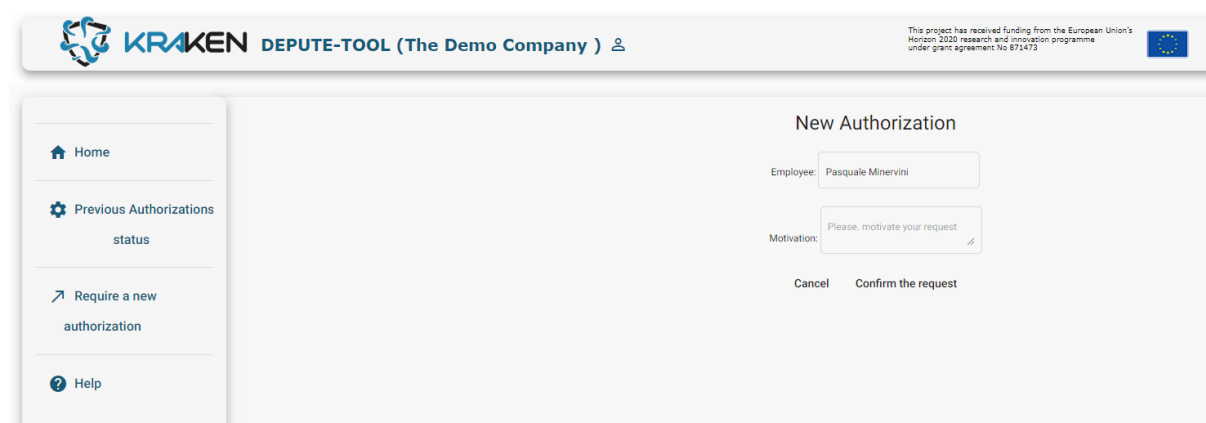


**Figure 24: Depute Tool Components**

## 15.2 Interfaces

The Depute Tool's software components expose the following interfaces:

1. Depute\_WebSite: A web user interface to manage The Depute Tool's authorizations. Figure 25 below shows a screenshot of the Depute Tool's user interface for an employee.



**Figure 25: Depute Tool Components**

2. DeputeExpressWebServer:
  - BackBackendProxy:
    - Acts as a proxy on the following Aries protocols:
      - DID Exchange
      - Issue-Credentials
    - Acts as a proxy on the MySQLAutorizazionDB api
  - WebhookListener: a private to KWCT exposed standard webhook listener (i.e., an http POST callback, according to a pattern which is widely used in the web development community) called by the LIM's Aries Go agent.
3. Aries\_GoRestAgent: standard Aries\_Go agent protocols used in KRAKEN [16][17] .
4. dbms:
  - agentDB: Private to LIM standard interface of CouchDB dbms.
5. dbms\_mysql:
  - MySQLAuthorizationDB: Private to depute standard interface of Mysql dbms.

## 15.3 Deployment

The Depute Tool instance is deployed as three containers in Kubernetes on AWS, these containers are described in [Figure 24](#):

- Depute\_ WebSite
- DeputeExpressWebServer
- Aries\_GoRestAgent

## 15.4 Source Code

The source code of these components can be found in the following private KRAKEN GitHub repositories:

<https://github.com/krakenh2020/DeputeAngularFrontEnd>

[https://github.com/krakenh2020/Depute\\_HttpProxy](https://github.com/krakenh2020/Depute_HttpProxy)

## 16 KRAKEN Company Identification Tool

The KRAKEN Company Identification Tool is one of the two software tools developed in KRAKEN to manage natural persons acting on behalf of a company. The tool manages the active list of companies that are able to delegate their employees to operate on behalf of the company they work for.

### 16.1 Description

The KRAKEN Company Identification Tool's architecture is organized in three tiers:

- **KCIT\_AdminWebSite:** An angular 13 web tool dedicated to KRAKEN's platform administrator users, used to populate the KCIT's database. Access to the tool is protected by KeyCloak using open-id-connect. The tool invokes the TIR\_Backend services through KCIT\_private interface.
- **KCIT\_backend:** A server-side component that exposes two REST API:
  - **KCIT\_Private\_CRUD\_api:** protected by KeyKloack and accessible only by the KCIT\_AdminWebSite, it provides Create, Read, Update and Delete (CRUD) functionalities on the KCIT\_database entities.
  - **KCIT\_public\_Api:** published to internet, provides methods to be used by the Marketplace.
- **KCIT database:** is the container of the configuration information of the KRAKEN Company Identification Tool.

Figure 26 below shows the KRAKEN Company Identification Tool components described in this Section.

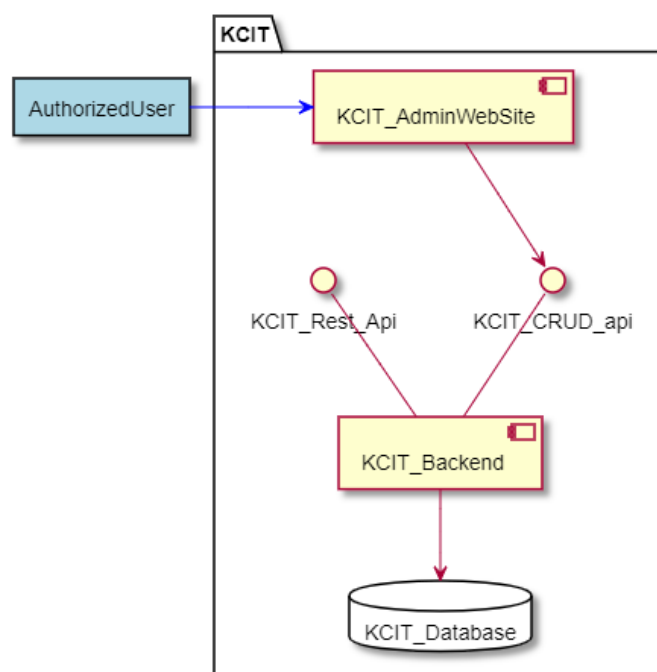


Figure 26: KCIT Components

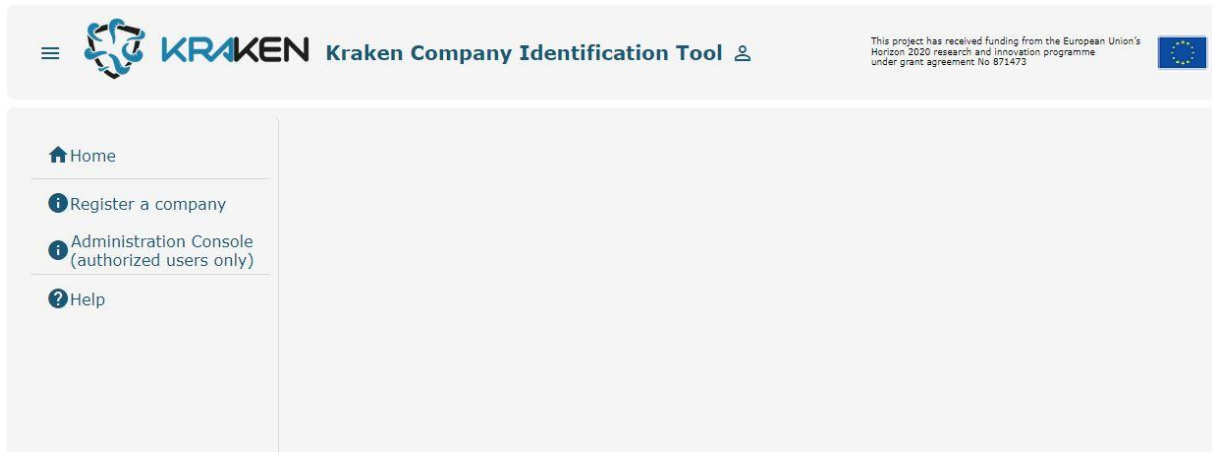
## 16.2 Interfaces

The described components expose the following interfaces:

- KCIT\_AdminWebSite: A private technical web user interface added to manage the KCIT's data.

The interface, after user authentication, offers the functionalities to manage the list of configured companies that are able to delegate an employee to operate on behalf of themselves.

Figure 27 below shows a screenshot of the private KRAKEN Company Identification Tool web interface.



**Figure 27: KCIT web interface**

KCIT\_CRUD\_API:

Not documented here because it is used only by the KCITAdmin website.

KCIT\_Rest\_Api:

- GET/kcit/activeCompanies  
Return the list of the companies authorized to delegate their employees
- GET/kcit/activeCompanies/did/  
Return the configured company that has the did passed as parameter

## 16.3 Deployment

The KCIT tool will be deployed as two docker containers as represented in [Figure 26](#).

## 16.4 Source Code

The source code of these components can be found in the following private KRAKEN GitHub repositories:

[https://github.com/krakenh2020/KCIT\\_angularFrontEnd](https://github.com/krakenh2020/KCIT_angularFrontEnd)

[https://github.com/krakenh2020/KCIT\\_httpProxy](https://github.com/krakenh2020/KCIT_httpProxy)





## 17 KRAKEN Revocation & Endorsement Registry

The KRAKEN Revocation & Endorsement Registry (KRER) was developed in the context of WP3 within T3.2 Universal Ledger Resolver activities. As a detailed information of this components has been generated in D3.2 Self-Sovereign Identity Solution Final Release, this document provides a general overview of the component and the necessary information to be integrated with the marketplace. Due to the Revocation & Endorsement Ledger will be released as a future service in the European Blockchain Services Infrastructure (EBSI) V2+ architecture, the KRAKEN Consortium decided to implement a microservice (KRER) mimicking its use. The development of KRER was based on the documentation provided at this moment by EBSI:

### 17.1 Description

The KRER component stores information related to the validity of the verifiable credentials (or Verifiable Presentations), such as the status (valid, revoked, suspend). Basically, the KRER comprises a core module which is offering the REST API services and the credential status storage.

### 17.2 Interfaces

The KRER offers 3 interfaces:

- credential/add (POST): Creates the status of a new credential.
- credential/update (PUT): Updates the status of an existing credential.
- credential/status (GET): Public interface for providing the credential status.

### 17.3 Deployment

The KRER has been developed to be deployed by the VC issuer, which keeps the control and manage the status of the credentials. With this aim a docker image is provided. Only the configuration of the open ports and the allowed IPs will be needed.

### 17.4 Source Code

As indicated in D3.1 the source code of this component can be found in the following private KRAKEN GitLab repository:

[https://scm.atosresearch.eu/ari/ledger\\_uself/ssi-ledgeruself-krer](https://scm.atosresearch.eu/ari/ledger_uself/ssi-ledgeruself-krer)

## 18 Privacy Metrics Tool

The Privacy Metrics Tool produced by AIT has been integrated into the KRAKEN Marketplace Frontend codebase and is described in this Section of the report. The Tool is applicable to data subjects that are publishing batch Data Products for direct data download and quantifies the level of privacy that data subjects can expect when sharing their batch Data Products within the marketplace. Details on the privacy metrics can be found in the [Annex](#) of this deliverable.

### 18.1 Description

The computation of the privacy metrics is performed in three steps. First, the user is asked to complete a questionnaire on some general facts that are not dependent on the data set that is sold. We note that in the current integration this questionnaire is always presented to the user to avoid storing data on a service of the marketplace. Alternatively, the data could be stored on the user's device which is currently not implemented. Second, the user is asked about dataset specific data. This questionnaire involves a set of questions on the types of data included in the data sets, such as educational data, medical data, and other potentially identifiable data. Last, after completing both questionnaires, the privacy metric is computed and presented to the user a keen to password strength indicators.

#### 18.1.1Description

The first questionnaire contains questions on dataset independent inputs that help to adjust the privacy metrics based on the user's self-assessment on the importance of privacy. It also takes into account a user's background and perceived threats. The questionnaire is depicted in [Figure 28](#).

**For estimating your privacy concerns please answer the following questions:**

Do you use social networks (like Facebook, LinkedIn, Twitter,...)? ▼

How present are you in public through your activities and your job? ▼

Who could be a potential adversary? (multiple answers possible) ▼

How many inhabitants has the city where you live? ▼

Please select the sector which describes your job best ▼

Are you active in an NGO? ▼

**How serious do you take your privacy in general?**

not at all very serious

**How important do you think is it to protect your health data?**

not at all very important

**How important do you think it is to protect your place of residence?**

not at all very important

**How strong do you want to protect information about your job or degree?**

not at all very secure

**SUBMIT**

**Figure 28: Dataset independent inputs to compute the privacy metrics**

### 18.1.2 Dataset Independent Inputs

In the next step, the user is asked to provide some metadata on the datasets the user intends to sell on the marketplace. For the privacy metrics relevant data points include information on the country and city of residence, respectively, Global Positioning System (GPS) location data, personal data such as age, birthday, gender, body height and weight, or other health data. Additionally, the number of data sets with similar data points also has an impact on the privacy metric and is hence taken into account. The full questionnaire is depicted in Figure 29 below.

### Calculate privacy metric

Does this file have similar information or does it have relating data to previously uploaded files? Please select all items that are contained in the file:

- ☐ Country
- ☐ City
- ☐ GPS-Location
- ☐ Age/Birthdate
- ☐ Gender
- ☐ Body size/weight
- ☐ Health data
- ☐ Profession/Job
- ☐ Degree/Diploma
- ☐ Connection to other people
- ☐ Grades
- ☐ Sports/physical activity
- ☐ Other data

Number of files

**1**

---

Number of files with counties

**0**

---

Number of files with cities

**0**

---

Number of files with GPS data

**0**

---

Number of files with age/birthdate

**0**

---

Number of files with gender

**0**

---

Number of files with height/weight data

**0**

---

Number of files with health data

**0**

---

Number of files with job/professional data

**0**

---

Number of files with degrees

**0**

---

Number of files with social connections

**0**

---

Number of files with grades

**0**

---

Number of files with sports data

**0**

---

Number of files other data

**0**

---

**CALCULATE PRIVACY**

Figure 29: Dataset dependent inputs for the privacy metrics

### 18.1.3 Privacy Value

Finally, the computed privacy metric is displayed to the user. Similar to password strength indicators, the colored bar is displayed where red indicates a high risk to a user's privacy whereas a green colored bar indicates a low risk. An example of this bar is presented in Figure 30 below.

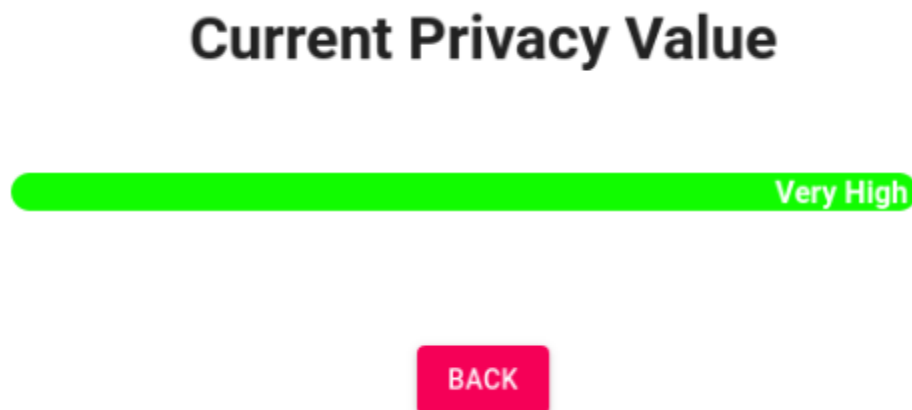


Figure 30: Example result from the privacy metrics computation

## 18.2 Interfaces

The privacy metrics widget provides three React components:

- **AdversaryStrength**: this component displays a questionnaire on dataset independent inputs for the computation of the metrics.
- **Input**: this component displays a questionnaire on the dataset dependent inputs for the computation of the metrics.
- **PrivacyVal**: this component displays the result of the privacy metrics computation based on the inputs from the other two components to the user.

The integration into the KRAKEN marketplace uses all three components to request the necessary inputs and to present the result to the user.

## 18.3 Deployment

The privacy metrics are integrated into user interface of the KRAKEN marketplace. Therefore, the components are deployed with the same process as the KRAKEN marketplace user interface. For test purposes, a development webserver can be started via npm.

## 18.4 Source Code

The full source code is publicly available at <https://github.com/krakenh2020/kraken-privacy-metrics>. The code is released under the Berkeley Software Distribution (BSD)-3-clause license to promote its use beyond the integration of the privacy metrics widget into the KRAKEN marketplace.

## 18.5 Baseline Technology and Tools

The main goal for the development of the privacy metrics widget is that none of the data required for the computation leaves the user's device. Therefore, the widget was designed to perform all computations on the user's device. For web-based services such as the KRAKEN marketplace, Javascript is a natural choice to accomplish this goal. Hence, the privacy metrics widget and all the computations have been developed with Javascript and the user interfaces employs the [React framework](#).

## 19 Conclusion

This D5.6 deliverable document describes the software of the final KRAKEN marketplace release. Over the course of this project, through the integration of multiple partners and third-party technologies at varying levels of technology readiness, a fully functional marketplace has been established which will be subsequently tested in the education and health pilots due to take place in September 2022.

The first marketplace release delivered as part of D5.5 in September 2021 included functionality that allowed users to exchange and monetise their data using the encrypted batch data transfer through directory sharing modality. It included an initial integration with the uSelf Broker of the SSI Agent, allowing users to register and login to the marketplace using the SSI mobile app and an initial integration with the MPC network, which provided a mechanism for key distribution in the batch data sharing modality. It also included an integration with the Lynkeus' blockchain, to provide a blockchain-based permissioning layer and the xDai blockchain for handling payment transactions between data providers and data consumers.

This conclusion summarises the major achievements in the software development for this final release. Since the initial release, time has been spent expanding the data exchange modalities available in the KRAKEN marketplace. In addition to the already available batch data exchange modality, new modalities have been developed and integrated that allow users to also create Data Products for privacy preserving analytics and Data Unions.

Analytics Data Products, realised through an innovative infrastructure that integrates blockchain-based permissioning, token-based payments and distributed computations, allow marketplace users to perform privacy-preserving analytics on datasets in a CSV format. Within this infrastructure, an MPC network computes standard statistics using functions such as average, standard deviation and min and max values. The marketplace allows these functions to be computed on multiple Data Products containing matching variables in a single data buyer transaction. Functionality has also been developed to share revenues in cryptocurrency between each owner of the Data Products used in the computation, and to retain a fee for the marketplace in return for the provision of this service.

Through the additional marketplace integration with the Streamr Network and Streamr's Data Union Framework, the final release also allows the publishing of a Data Union Data Product in the KRAKEN marketplace. Data Unions consist of multiple persons, or even entities, opting in with the use of a third-party application to share and monetise data streams in a single Data Product that is managed and administered by a Data Union Manager. An example of this could be a mobile app that allows multiple individuals with health or fitness data being generated on their mobile phones to opt-in to a Data Union. Data streams are transported from data providers to data consumers using the decentralized Streamr Network, and revenues in cryptocurrency generated when access to the Data Union sells within the KRAKEN marketplace are shared between all members of the Data Union. A set percentage of the revenue can also be directed to the Data Union Manager in return for their services.

As well as the additional data exchange modalities realised within this final release, other major achievements include the further development of the marketplace user registration and login functionalities. Within this release, through the integration with the uSelf Broker of the SSI Agent, KCIT and Depute Tool, a legal person working on behalf of a company or institution to purchase or sell data within the marketplace can register by presenting a proof of their Attorney Credentials that represent their company affiliation. Within this process the marketplace also verifies the issuer (company) and verifies that the credential has not been revoked.

Finally, a KRAKEN Marketplace Mobile app has also been developed and integrated with the Marketplace Backend to allow users to quickly browse Data Products, change permissions and availability of their own Data Products and see how well their Data Products are performing on the market.

## 20 References

- [1] [KRAKEN Project: D5.5 Marketplace Initial Release, 2021](#)
- [2] [KRAKEN Project: D2.7 Design for Marketplace Reference Implementations, 2022](#)
- [3] [KRAKEN Project: D2.6 Marketplace Technical Specification, 2020](#)
- [4] [KRAKEN Project: D2.3 Final KRAKEN Architecture, 2021](#)
- [5] [KRAKEN Project: D3.2 Self Sovereign Identity Solution Final Release, 2022](#)
- [6] <https://streamr.network/discover/data-unions/>
- [7] <https://streamr.network/core>
- [8] <https://streamr.network/docs/streamr-network/intro-to-streamr-network>
- [9] <https://github.com/streamr-dev/streamr-specs/blob/master/PROTOCOL.md>
- [10] [KRAKEN Project: D4.4 Final Implementation of Cryptographic Libraries, 2022](#)
- [11] [KRAKEN Project: D5.3 Initial KRAKEN Marketplace Integrated Architecture, 2021](#)
- [12] [KRAKEN Project: D2.5 KRAKEN Final Technical Design, 2022](#)
- [13] <https://keycloak.org/>
- [14] <https://openid.net/connect/>
- [15] [Hyperledger Aries Framework Go: https://github.com/Hyperledger/aries-framework-go](https://github.com/Hyperledger/aries-framework-go)
- [16] [Hyperledger Aries RFC 0023: DID Exchange Protocol 1.0: https://github.com/Hyperledger/aries-rfcs/tree/main/features/0023-did-exchange](https://github.com/Hyperledger/aries-rfcs/tree/main/features/0023-did-exchange)
- [17] [Hyperledger Aries RFC 0453: Issue Credential Protocol 2.0: https://github.com/hyperledger/aries-rfcs/tree/main/features/0453-issue-credential-v2](https://github.com/hyperledger/aries-rfcs/tree/main/features/0453-issue-credential-v2)



## 21 Annex 1: Selecting Privacy Metrics for KRAKEN

### 1. Introduction

KRAKEN is a data market for privacy-sensitive data and “aims to enable the sharing, brokerage, and trading of potentially sensitive personal data, by returning the control of this data to citizens throughout the entire data lifecycle” [2]. In other words, KRAKEN is a platform that facilitates the transfer of data and offers its users a high degree of privacy, which is defined as “the ability of an individual to control the terms under which personal information is acquired and used” [1]. In order to give users insights into their privacy while uploading information to this database, KRAKEN’s objective is to utilize privacy metrics to measure the users’ privacy and provide them with an overview of how well it is protected.

A privacy metric is defined as the “degree of privacy enjoyed by users in a system and the amount of protection offered by privacy-enhancing technologies” [2]. More actively worded, privacy metrics are “measures to determine the susceptibility of data or a dataset to revealing private information” [1]. Privacy metrics try to quantify the level of privacy in a system or the privacy provided by a privacy-enhancing technology (PET). Subsequently, this Section attempts to identify the right metrics to measure KRAKEN’s level of privacy.

In order to select the most fitting metrics for KRAKEN, this Section will consist of eight subsections that try to illustrate and explain how KRAKEN could measure the user’s or data subject’s privacy and why the chosen metrics are an efficient indicator to quantify privacy. Therefore, the first few parts will primarily explain and illustrate the playground for privacy metrics by examining existing papers and research. Whereas the second half of the paper intends to apply those concepts to the KRAKEN database and identify metrics, as well as give an explanation on how a potential quantification of privacy might look.

Subsequently, the first part of this Section will discuss the potential output measures for KRAKEN, which helps to identify the best privacy metric categories for KRAKEN. The second part focuses on the adversary model and specifies the different adversary capabilities that KRAKEN needs to consider. The third part gives an overview of the type of data that KRAKEN tries to protect by examining the Use Cases from the KRAKEN deliverables. Furthermore, this Section will also elaborate on linkage attacks and data that might appear to be anonymous but still has the ability to harm a user’s privacy if it gets into the wrong hands. The fourth subsection talks about the different inputs needed to calculate the privacy metrics and explains how this database system could a final privacy result. Subsection five will explain which privacy metrics can be considered for KRAKEN and why they are more efficient than other metrics. The sixth part talks about the target audience for KRAKEN and how the privacy metrics should work and look from the user’s point of view. Part seven explains the recommended quantification of the KRAKEN privacy metrics and tries to give insights on why those measures are efficient and effective in attaining their goal in offering the user information about their privacy.

Lastly, this Section will offer recommendations on how privacy metrics could be implemented. Moreover, this subsection will discuss how the input for the privacy metrics can be obtained and how the output should look. Besides, part eight also offers pseudo-code to facilitate the implementation of the metric’s quantification.

### 2. Output Measures

The output of a privacy metric refers to the kind of property that the metric measures. Each output property represents a different privacy aspect. Subsequently, it is essential to note that a single metric cannot capture the entire concept of privacy. So, it is recommended to utilize metrics from various concepts to get a complete privacy estimate [1]. There are eight output categories, which were identified by Eckhoff and Wagner [1].

1. **Uncertainty:** Uncertainty metrics are based on the assumption that the system's privacy is high if the uncertainty of the adversary's estimates is high.
2. **Information gain or loss:** In relation to information theory, the amount of privacy lost based on the disclosure of information or the amount of information gained by the adversary is measured by privacy metrics focusing on information gain or loss.
3. **Data similarity:** Data similarity metrics measure the similarity of data either within a dataset or between two sets of data. In this way, these metrics abstract from the adversary and focus on the properties of data.
4. **Indistinguishability:** Metrics based on indistinguishability analyze the various outcomes of the privacy mechanisms. If the adversary cannot distinguish between any pairs of outcomes, privacy is considered high.
5. **Adversary's success probability:** Metrics using the success probability describe the likelihood of the adversary's attempt to reveal privacy. In this way, low success probability correlates with high levels of privacy.
6. **Error:** Error-based metrics measure the correctness of the adversary's estimate, i.e., the distance between the correct outcome and the prior estimate. In this context, high correctness directly relates to low privacy levels.
7. **Time:** Time-based metrics consider the time until the adversary succeeds in breaking the privacy of the system. Longer times correlate to higher privacy levels.
8. **Accuracy or precision:** Some privacy metrics quantify how precise the adversary's estimates are. More accurate estimates correspond to lower privacy.

Since KRAKEN provides a new PET, the focus should be on displaying the efficacy of this project. As a result, the best metric categories for this project are accuracy, similarity, and indistinguishability. The reason is that metrics in the categories, time, error, and adversary's success probability, have a higher focus on the adversary, whereas the other three categories concentrate on the efficacy of the PET, which is a more suitable approach for a project such as KRAKEN.

Nonetheless, it is also beneficial to include metrics that focus more on the adversary, such as time, error, or the adversary's success probability, to capture the entire concept of privacy. This approach would allow KRAKEN to get a complete and compact estimate of privacy.

### 3. Adversary Model

The adversary's goal is to compromise users' privacy and learn sensitive information, such as user identities and user properties [1]. To pick the most relevant metrics for KRAKEN, one must know the adversary's capabilities, such as its knowledge or resources. Furthermore, it is vital to identify the adversary's goals and what type of information it is trying to get. LINDUDUN, a threat modeling methodology for systematically analyzing privacy threats in software architectures, identified ten major threats by analyzing the KRAKEN architecture [2]. These threats are shown in Table 1 below and will be utilized to determine the adversary's fitting capabilities and goals. The privacy threats that are marked green have a low likelihood, impact, and priority, and therefore, they will not be considered.

Threat	Likelihood	Impact	Priority
Linkability in one or more storages.	Medium	Medium	Medium
Identifiability in one or more storages.	Low	High	Medium
Detectability of data existence.	Medium	Low	Low
Detectability in communication between different trust domains.	Low	Low	Low
Linkability of IP addresses in communication between different trust domains.	Low	Medium	Low
Linkability of IP addresses in communication between different trust domains leads to identifiability.	Low	High	Medium
Non-repudiation of encrypted data.	Low	Low	Low
Non-repudiation of communication between different trust domains.	Low	Low	Low
Unawareness of data owner.	Low	High	Medium
Non-deletion of data in cloud storage.	low	Low	Low

**Table 1, Threat Table, Source: Privacy-preserving Analytics for Data Markets using MPC [2]**

Table 1 displays the different KRAKEN threats and shows the likelihood, impact, and priority for each threat. The range for each attribute is low, medium, and high, where low values are colored green, medium values are colored yellow, and high values are colored red. According to Table 1, the first threats that this Section will discuss are the ones that have a medium priority. *Linkability in one or more storages* is a threat in which the adversary is an insider who exploits storages to raise his or her chances of linking the user's data. Similarly, *identifiability in ore more storages* is a threat that focuses on an insider adversary. The adversary has access to the data storages and tries to obtain a set of information that can be linked and ultimately identify one or more users. However, the adversary would also need additional information other than the one within KRAKEN [2].

The third threat focuses on the *linkability of Internet Protocol (IP) addresses* in communication between different trust domains leads to identifiability. This threat focuses on an external or internal adversary who can access the user's network and inspect the user's packets. By doing that, the adversary can link the IP address to the user's identity. The last threat with a medium priority is the *unawareness of the data owner*. There exists no adversary in this scenario since the data owner is making his or her data available. For scenarios like this, the best privacy metrics are indistinguishability metrics because this is the only category that does not focus on an adversary but the data's properties [2].

The last two threats discussed have a low priority but have either a medium impact or medium likelihood. *Detectability of data existence* is a threat in which the user uploads data on the cloud without publishing it on KRAKEN. That would allow an external adversary to find out about the existence of the data. Lastly, *the linkability of IP addresses in communication between different trust domains* includes an internal or external adversary who manages to link different events to the same

user by listening to the user's requests. In order to be capable of doing that, the adversary must be skilled and have access to the user's network [2].

Threat	Local - Global	Passive - Active	External - Internal	Prior knowledge
Linkability in one or more storages	Global	Active	Internal	Not necessary
Identifiability in one or more storages	Global	Active	Internal	Not necessary
Detectability of data existence	Local	Passive	External	Not necessary
Linkability of IP addresses in communication between different trust domains	Local or Global	Active	Internal or External	Necessary
Linkability of IP addresses in communication between different trust domains leads to identifiability	Local or Global	Active	Internal or External	Necessary
Unawareness of data owner	Local	Passive or Active	External	Not necessary

**Table 2, Threat characteristics**

As a result, the adversary's capabilities, which can be identified from the threats and must be considered for KRAKEN's privacy metrics, are as follows [1]:

- *Local - Global*: Local adversaries can only access parts of the system, whereas global adversaries can access the entire system.
- *Passive - Active*: Passive adversaries can only read or observe the system. Active adversaries can interfere and add, remove or modify information and use that to their advantage.
- *External - Internal*: External adversaries are not part of the system. Internal adversaries act from within the system, e.g., because they are working for service providers or third parties controlling specific components of the system.
- *Prior Knowledge*: The adversary has additional information about the system or has scenario-specific knowledge.

Furthermore, the adversary goal that results from the threats is to find out information about the user and get to know their identity.

## 4. Data Sources and Use Cases

Defying the KRAKEN's data source is essential because depending on which data source needs protection, different metrics apply. The KRAKEN-deliverable 2.6 gives insights into two pilot market sectors: health and education [3]. This part will discuss potential threats regarding the use cases by looking at incidents from the past and calling out similar scenarios within the use cases. First, this subsection will give information about the type of data within KRAKEN and the use cases. Second, it will discuss threats regarding this data by analyzing past events and reflecting on the use-cases.

First, the type of data within KRAKEN qualifies as published data. Published data "refers to information that has been willingly and persistently made available to the public" [1], e.g., statistical databases and information that individuals choose to disclose. If published data gets into the wrong hands, it turns into re-purposed data, which is defined as data used for different purposes than the purpose it was initially acquired [1]. One example of published data turning into re-purposed data is adversaries trying to identify anonymized individuals or reveal sensitive information using the user's published data, which was also the adversary's goal, as discussed in part 3.

In the past, there were multiple incidents where published data turned into re-purposed data. In most of these scenarios, the data that was misused seemed highly anonymous. But even if data appears to be anonymous, it still can reveal information to an adversary. Those incidents are known as linkage attacks, which happen when seemingly anonymous data are combined to reveal real identities [4]. An example of such data is the zip code, birth date, and gender of Americans. Although this published data appears to be anonymous, in combination, it is already enough information for an adversary to identify 87% of the American population [5]. Two past events that are famous examples of linkage attacks, in which an adversary revealed people's identities by giving away data that seemed anonymous, are the case of Governor William Weld [8][9] and the Netflix Prize [6][7].

The case of Governor William happened in 1996. It all started during a commencement ceremony at Bentley College, the Governor of Massachusetts collapsed, and he had to be taken to a hospital, where he received multiple diagnostic procedures. The next day, Wend was already discharged and recovered quickly [8]. The second event happened in the mid-1990ies, in which the Group Insurance Commission (GIC), a government agency, purchased health insurance for state employees and decided to release the records to any researcher who requested them. The records included every state employee's hospital visit at no cost. Like Netflix, they removed any direct identifiers to 'anonymize' the data. But it still had information such as ZIP code, birth date, or gender.

When the GIC released the data, Governor Weld supported their action and claimed that the patient's privacy was protected since the identifiers were deleted. As a result, an MIT graduate student, Latanya Sweeney, got involved and tried to find Governor Weld's hospital data in the GIC records. In addition to the GIC data, Sweeney bought the voter rolls from the city of Cambridge, which is Weld's hometown and has a population of 54,000. Then, by comparing the two datasets, it was easy for Sweeney to identify Weld because within Cambridge, only six people had the same birth date, only three were men, and only one lived in the same ZIP code. Therefore, Sweeney showed that it was easy to re-identify people through the GIC records through reverse-engineering [9].

The case of the Netflix Prize happened in 2006. Netflix created a competition known as the *Netflix Prize*. The competition's goal was to develop a recommendation system that was 10% more accurate than the previous one. Therefore, Netflix published a dataset that contained 10 million movie ratings, including the name of the movie and the date of the rating, by almost 500,000 customers so that people could work on this task. But to make the data anonymous, Netflix changed the users' names to unique IDs [6]. Netflix was convinced that this would already be enough to protect the data, and when they were asked if there is any data in the dataset that should be kept private, Netflix gave the following answer:

"No, all customer identifying information has been removed; all that remains are ratings and dates. [...] Even if, for example, you knew all your own ratings and their dates, you probably couldn't identify them reliably in the data because only a small sample was included (less than one-tenth of our complete dataset), and that data was subject to perturbation. Of course, since you know all your own ratings, that really isn't a privacy problem, is it?" [7].

But, according to Narayanan and Shmatikov, "removing the identifying information from the data is not sufficient for anonymity. The attacker may be able to join the (ostensibly) anonymized dataset with auxiliary data, resulting in a complete breach of privacy", which was the case with the Netflix Prize dataset. Narayanan and Shmatikov managed to de-anonymize some of the Netflix data and identify two users by comparing the rankings and timestamps with public information on IMDb. That allowed them to show how little information is needed to breach a person's privacy [6] [7].

The scenarios above showcase the dangers of underestimating data and illustrate what can happen if published data is misused and turned into re-purposed data. Consequently, it is crucial to pay attention to such scenarios within the KRAKEN database and be pro-active towards potential linkage attacks.

The first KRAKEN use-case that gets described in Deliverable 2.6 [3] focuses on health data. Consequently, 4.1 gives an overview of different types of information that fall in the category of health data and have the potential to lead to linkage attacks.

## 4.1 Health Data

- *Heart rate*: The heart rate can give away a lot of information about a person, such as their health situation or fitness level. For example, the adversary would be able to see if the user has a slow, fast, or normal health rate. As a result, an adversary could make assumptions about the user and their condition by accessing this information. Therefore, the adversary could increase their chances of identifying the user, lowering the user's privacy.
- *Prescriptions*: Information about a user's prescriptions can benefit the adversary since they would be able to see what kind of medication a user receives or even what type of disease the user has. That would help the adversary close down the circle of potential identities for this document and increase his chances of succeeding, negatively affecting the user's privacy.
- *Check-in & Check-out date*: The user's check-in and check-out date inform an adversary about the timeframe in which the user was in a hospital. As a result, the user could try to find out by using tools such as Facebook, which people were in the hospital during this time and close down the number of potential identities. Therefore, this type of information harms the user's privacy.
- *Patient's doctor in charge*: Information about the patient's or user's doctor in charge would tell the adversary, which station the user is in, what their potential disease/problem is, and even what type of insurance the user has, for example, if it is a doctor that only treats clients with private *insurance*. Therefore, figuring out this information would have a negative impact on the user's privacy.
- *Laboratory/test results*: The user's laboratory and test results could provide the adversary with all types of information about the user, such as blood type, diseases, or allergies. Consequently, it would facilitate the adversary's goal of identifying the user, which lowers the user's privacy.
- *Diagnosis*: The user's diagnosis gives away information about the user's disease/injury/etc. Therefore, it makes it easier for the adversary to figure out their identity. That leads to a negative impact on the user's privacy.
- *Therapy sessions*: Information about the user's therapy sessions would give away details about the user's treatment and injury/disease/etc. Furthermore, it might even inform the adversary about the time and location of the therapy. That would close down the circle of potential identities and, therefore, lower the user's privacy.
- *Income information*: Income information lets the adversary know to which income class the user belongs. That would reduce the number of potential identities and raise the chances of identifying the user.
- *Insurance company/information*: Information about the insurance company or type of insurance gives away information such as where the user is from and if the user has public or private insurance. That would make it easier for the adversary to identify the user since they can restrict the number of potential identities.

The second KRAKEN use-case is education data. Subsequently, 4.2 gives an overview of different types of information that fall in the category of *education data* and have the potential to lead to linkage attacks.

## 4.2 Education Data

- *Class ranks*: Class ranks show in which percentile the user is. For example, the user is in the top 10%. Therefore, the adversary could improve their chances of identifying the user by using the class rank combined with additional documents or information the adversary found out. Consequently, uploading the user's class ranks would reduce their privacy.



- *Class enrollment:* The class enrollment tells what classes a user is currently enrolled in and which classes a user took in the previous academic years. Consequently, the adversary could assume the user's major and minor, as well as how far the user is in their studies and when they started their academic career, which would have a negative impact on the user's privacy since it makes it easier for the adversary to figure out the user's identity.
- *Scholarship information:* Information about the user's scholarship allows the adversary to find additional information about a user. For example, the scholarship could tell the adversary if the user belongs to a minority, their gender, etc. Subsequently, it would raise the adversary's chances of finding out the user's identity and lower the user's privacy.
- *Graduation date:* The graduation date would allow the user to narrow down the number of possible identities because universities have public numbers about how many students graduated in an academic year. Therefore, the adversary would know how many potential identities there are, and they would be able to identify the user by using additional resources and information. Therefore, this type of information has a negative impact on the user's privacy.
- *Enrollment date:* The enrollment date would allow the user to narrow down the number of potential identities because universities have public numbers about how many students enrolled in an academic year. Therefore, the adversary would know how many possible identities there are, and they would be able to identify the user by using additional resources and information. Consequently, this type of data might lower the user's privacy.
- *Campus jobs:* Information about a user's campus jobs can help the adversary increase their chances of identifying the user. For example, suppose a document contains the information that the user has a particular campus job. In that case, the adversary could either find the person on the university's website or look up web pages, such as LinkedIn, to identify the user. As a result, uploading a user's campus job would hurt their privacy.
- *Participation records:* Participation records, such as information on which student organizations a student is a part of, allows an adversary to find out the user's identity easily. Most student clubs and organizations have public web pages that show their members. If an adversary manages to obtain this information, they can look up the people in those clubs and organizations and then identify the user. Subsequently, this would have a significant impact on the user's privacy.
- *Meal plan:* Meal Plans can give away a high amount of information about a user, such as their housing situation and school year. For example, universities in the USA offer different meal plan options depending on what year you are and your housing situation. For instance, if a person is a sophomore and lives off-campus, this person will receive a different meal plan than a freshman who lives on-campus. Therefore, an adversary can close down the circle of potential identities by obtaining this type of information, which would negatively impact the user's privacy.
- *GPA (Grade Point Average):* GPA gives information about a user's academic success and allows the adversary to assume which class percentile the user is. Consequently, it would lower the number of potential identities, which would reduce the user's privacy.

Those are just a few examples of inconspicuous data. Even though it doesn't seem significant on its own, but as the case of Governor Wend and the Netflix Prize showed, it is necessary to pay attention to data even if it appears to be anonymous.

## 5. Input Data

There exist five input categories one can use to calculate privacy metrics. Depending on the chosen metrics, different inputs apply. Therefore, it is crucial to pay attention to which inputs one has available before deciding what privacy metrics will be implemented. The five input categories, according to Eckhoff and Wagner, are as follows [1]:

- *Adversary's estimate*: The adversary's estimate results from the adversary's effort to breach privacy.
- *Adversary's resources*: The resources that are available to the adversary.
- *True outcome*: The true outcome is used to judge how good the adversary's estimate is. But this information is not available to the adversary, so they can't compute metrics that use true outcome.
- *Prior knowledge*: Prior Knowledge describes concrete, scenario-specific knowledge that the adversary has.
- *Parameters*: Parameters describe threshold values. The sensitivity of attributes, which attributes are sensitive or desired privacy levels.

The idea for the KRAKEN privacy metrics is to utilize metrics from multiple categories to get a privacy metric that is compact and offers detailed insights about a user's privacy. Therefore, if applicable, either of the above inputs might be used if they are needed to get a chosen metric. But since KRAKEN tries to offer privacy metrics that are simple and easy to understand, it is recommended to use the inputs from Table 3.

Adversary estimate	True outcome	Parameters
--------------------	--------------	------------

**Table 3, KRAKEN Input**

The idea to get those inputs is that some are received directly from the user, whereas the other inputs come from KRAKEN. For example, if a user wants to upload a document or file onto the KRAKEN cloud, they will need to fill out a form or give information that will translate into meta-data. This meta-data will then represent one of the inputs from Table 3 and calculate the privacy metrics. Furthermore, the user will also have to decide what type of adversary they want and need to protect their data. For example, some users might have less sensitive data to upload and therefore, are less concerned about the adversary's abilities and how it would affect their privacy.

In contrast, another user might upload highly sensitive data and needs to know how secure their privacy is should it be affected by a highly-skilled adversary. As a result, the user will have different adversary categories to pick from, which will be discussed in more detail later. The final result of the privacy metrics should be a combination of all the utilized metrics. Although, it depends on the chosen privacy metrics to determine what precisely the combination would look like. The broad idea behind it looks as follows and will be discussed in more detail in the Quantification section:

After the user gives KRAKEN the information about their data, the system will use the input to calculate the user's privacy. Each metric uses an interval that is separated into six sub-intervals which represent the different privacy levels. The six privacy categories can be seen in Table 4. For example, suppose the value for a privacy metric is high. In that case, it will belong to the "Very High"-privacy category, whereas, if it is low, it will belong to the "Very Low"-privacy category.

Each privacy result has its own value range, where the lowest category has the value 1, and the highest has the value 6. In the end, the metric values will be summed up and divided by the number of metrics to calculate an average privacy value. The average privacy value will then be compared to a grading system, as shown in Table 4, which gets the user's privacy level.



Privacy Category	Value Range
Very High	6 – 5.5
High	5.49 – 4.5
Moderately High	4.49 – 3.5
Moderately Low	3.49 – 2.5
Low	2.49 – 1.5
Very Low	> 1.5

**Table 4, Privacy category: Grading scheme**

Example #1 and Example #2 will give a brief and straightforward overview of how the calculation is supposed to work.

### 5.1 Example #1

Table 5 illustrates what the calculation of the user's privacy should look like. The column declared as *output category* shows to which output category the metrics from column two to column four belong, and the last column shows the results of each row. But before further explaining how the calculation works, it is essential to note that this calculation would work regardless of the number of metrics used.

For example, one could use only two output categories with one metric per category and still would receive a valid output. The important part is choosing the right metrics and utilizing them in a way that makes sense for the database. Furthermore, the following examples do not offer insight into the quantification of the metrics but rather explain how the chosen metrics and output categories could be combined to get one final privacy metric.

As can be seen in Table 5, the second and third column displays the metrics of *data similarity* and *indistinguishability*. Both categories consist of three metrics, whereas only two metrics are utilized from the other three output categories. That doesn't mean that one category is more efficient than another or that one category has more metrics than another. The only purpose of this example is to show how different metrics from different categories can be combined.

Metric 1 shows that it has the value 6, Metric 2 displays that it holds the value 4, and metric 3 has the value 5. Once the metrics got calculated and assigned to their values, the values of each output category will be summed up. Therefore, *data similarity* has a total of 15. The same procedure will be done for the four other output categories as well. Therefore, *indistinguishability* has a total of 12, *accuracy* sums up to 9, *error* is 5, and *adversary's success probability* holds 3.

Afterward, the totals will be summed up, which can be seen in row seven, leading to a total of 44. After that, the value of 44 will be divided by 12, the number of metrics used, to get an average privacy value for the calculated metrics. The final result is 3.67, and once the final result is calculated, the value gets compared to Table 4 and assigned to its privacy category. The user would have moderately high privacy in this scenario since the final result is between 4.49 and 3.5.

Output Category	Metric 1	Metric 2	Metric 3	Result
Data Similarity	Very High <input type="checkbox"/> 6	Moderately High <input type="checkbox"/> 4	High <input type="checkbox"/> 5	$6 + 4 + 5 = 15$
Indistinguishability	Moderately Low <input type="checkbox"/> 3	Moderately High <input type="checkbox"/> 4	High <input type="checkbox"/> 5	$3 + 4 + 5 = 12$
Accuracy	High <input type="checkbox"/> 5	Moderately High <input type="checkbox"/> 4	/	$5 + 4 = 9$
Error	Moderately Low <input type="checkbox"/> 3	Low <input type="checkbox"/> 2	/	$3 + 2 = 5$
Adversary's Success Probability	Low <input type="checkbox"/> 2	Very Low <input type="checkbox"/> 1	/	$2 + 1 = 3$
Add results	/	/	/	$15 + 12 + 9 + 5 + 3 = 44$
Divide by 12 (Divide by number of metrics)	/		/	$44 / 12 = 3.67$ <input type="checkbox"/> Moderately High

Table 5, Privacy metrics calculation: Example 1

## 5.2 Example #2

Table 6 illustrates a second example of how the metrics are combined to get the user's privacy. Once again, the column declared as *output category* shows to which output category the metrics from column two to column four belong, and the last column shows the results of each row. The second row displays the output category data similarity and its metrics. This time it only has one metric with low privacy. Therefore, the total for data similarity is 2.

The same procedure will be used for the other output categories. Subsequently, indistinguishability sums up to 11. Accuracy sums up to 12, error has a total of 15, and adversary's success probability has a total of 10. The summation of the output category results is 50. This value gets divided by the number of metrics that were used, which is 12. Consequently, the average privacy metric for this scenario is 4.55, which tells the user that their privacy is high since the result is between 4.5 and 5.5.

This time the output categories utilized a different number of metrics for each output category, but the final result calculation still worked the same. Subsequently, it should be evident that this approach works for any combination of output categories and metrics.

Output Category	Metric 1	Metric 2	Metric 3	Result
Data Similarity	Low □ 2	/	/	2
Indistinguishability	Very High □ 6	High □ 5	/	6 + 5 = 11
Accuracy	Moderately High □ 4	High □ 5	Moderately Low □ 3	4 + 5 + 3 = 12
Error	High □ 5	Very High □ 6	Moderately High □ 4	5 + 6 + 4 = 15
Adversary's Success Probability	Very High □ 6	Moderately High □ 4	/	6 + 4 = 10
Add results	/	/	/	2 + 17 + 9 + 11 + 10 =
Divide by 12 (Divide by number of metrics)	/	/	/	50 / 11 = 4.55 □ High

Table 6, Privacy metrics calculation: Example 2

## 6. Potential KRAKEN Privacy Metrics

Output Categories	Adversary Characteristics	Available Input Data	Data Sources
Accuracy	Local – Global	Parameters	Published Data
Similarity	Passive – Active	Adversary Estimate	Re-purposed data
Indistinguishability	External – Internal	True Outcome	
Uncertainty	Prior knowledge		
Information Gain			

Table 7, KRAKEN's characteristics for privacy metrics

Table 7 summarizes the different attributes, characteristics, and categories defined in the earlier parts of this Section. The first column shows the main output categories that should be used for the privacy metrics. The first three categories were chosen because they give a good indicator of KRAKEN's efficacy. But as mentioned earlier, it is also recommended to use a few metrics from either time, information gain, uncertainty, or adversary's success probability to get a complete estimate of privacy in KRAKEN.

Column two displays the different adversary characteristics which were identified through the KRAKEN threats. Some of them oppose each other, such as local and global or passive and active. However, it is still important to pay attention to either characteristic since an adversary can have a different characteristic, depending on the threat scenario. For example, linkability in one or more storages focuses on an internal adversary, whereas unawareness of data owners concentrates on external adversaries.

Column three shows the different input data that will be needed to get the privacy metrics. Depending on the chosen metrics, different inputs apply. For KRAKEN, we identified three particular inputs to be the most reliable since they allow us to keep the privacy metrics simple. Those inputs are adversary estimate, true outcome, and parameters. Through those inputs, it should be possible to get a compact privacy metric and inform the user how low or high their level of privacy is within KRAKEN.

The last column summarizes the two types of data within the KRAKEN database. First, most of the data will consist of published data, which is described as “information that has been willingly and persistently made available to the public” [1]. Second, the data can turn into re-published data, which is defined as information used for a different purpose than intended.

## 7. Target Audience

Since KRAKEN’s target audience will consist of many laypeople, it is beneficial to select privacy metrics that are easy to understand. Therefore, each privacy metric must have an intuitive interpretation. A potential consideration would be implementing a universal “translation” of the privacy metrics that allow the users to understand how safe their privacy is. For example, the American grading system (Letter Grades: A, B, C, D, E, F) could be an option to display the user’s privacy. Another option would be using a more common visualization and depiction of privacy metrics. Many online services that require the user to create a password have an indicator that lets them know how safe their password is. The safety degree in these cases ranges from “Weakest” to “Strongest,” as shown in Figure 1.



Figure 1, Practical Example of Privacy Metrics depiction, Source: OSU ONID

Furthermore, it is also vital to make it easy for the user to give KRAKEN information about their data and potential adversaries. As discussed in Section 5, the user will need to provide additional information about the document or file they want to upload to create meta-data for the privacy metrics and decide what type of adversary they need protection against. First, the adversary categories for KRAKEN can be separated into five different qualitative values, according to the National Institute of Standards and Technology (NIST). The five adversary capability values are [10]:

- *Very high*: The adversary has a very sophisticated level of expertise, is well-resourced, and can generate opportunities to support multiple successful, continuous, and coordinated attacks.
- *High*: The adversary has a sophisticated level of expertise, with significant resources and opportunities to support multiple successful coordinated attacks.
- *Moderate*: The adversary has moderate resources, expertise, and opportunities to support multiple successful attacks.
- *Low*: The adversary has limited resources, expertise, and opportunities to support a successful attack.
- *Very Low*: The adversary has minimal resources, expertise, and opportunities to support a successful attack.

Even though NIST mentions five categories to represent an adversary's capability, it is recommended to decide on an even number of capability values to eliminate the middle choice. Therefore, it might be beneficial to split it into Moderately High and Moderately Low to choose from an even number of values. That would leave KRAKEN with six different categories to represent the user's privacy and the adversary's capabilities. The six categories are as follows:

Very Low	Low	Moderately Low	Moderately High	High	Very High
----------	-----	----------------	-----------------	------	-----------

**Table 8, Privacy categories for KRAKEN**

## 8. Quantification

As discussed earlier, the metrics' goal is to be simple and to give the user a brief overview of how their privacy in KRAKEN is affected by uploading or downloading a file. Consequently, this Section tries to propose potential metrics that are derived from the metrics discussed by Eckhoff and Wagner and illustrate how KRAKEN might use them in a practical setting. Before going into more detail on how those metrics might look, it should be said that most of the metrics that were identified in other papers did not aim to give a user a particular degree or value of their privacy. Those metrics try to show and explain how one can make their data more private by utilizing methods that make the data more secure.

For example, k-anonymity, one of the privacy metrics from [1], is a disclosure technique for releasing information from a private table so that the identity of individuals to whom the data refers cannot be definitively recognized [11]. As a result, it is challenging to utilize most of the metrics to satisfy KRAKEN's objective since many of the metrics are hard to transform and implement to benefit the KRAKEN project. Nevertheless, three metrics were identified that could give the user insights on how and why their privacy is negatively affected by uploading and downloading files to KRAKEN and informing them how they might increase their privacy again.

### 8.1 KRAKEN Metrics

The chosen privacy metrics are based on the two methods, differential privacy and confidence interval width, discussed by Eckhoff and Wagner [1]. One should mention that they only have little in common with the original theories and only try to grasp the idea behind those metrics. Furthermore, it is also important to know that these metrics are only recommendations. One can implement additional metrics or even change the metrics if they identified metrics that fit their purpose better.

Regardless of the chosen metrics, one can utilize the basic idea of this quantification, but for reasons that will be discussed later, we decided to focus on the following three privacy metrics.

#### 8.1.1 Metric #1: Confidence Interval Width

The confidence interval width says that if "it can be estimated with  $c\%$  confidence that a value  $x$  lies in the interval  $[x_1, x_2]$ , then the interval width  $(x_2 - x_1)$  defines the amount of privacy at  $c\%$  confidence level [12]. As a result, the formula for the confidence interval width looks as follows [1]:

$$\text{privCIW} \equiv |x_2 - x_1| \text{ where } p(x_1 \leq x < x_2) = c / 100$$

Since this metric tries to identify the estimate that a specific value  $x$  lies in an interval, KRAKEN will define a new simplistic metric that solely based on the definition of Confidence Interval Width. For KRAKEN, the idea is to use the confidence interval width to get the number of files an adversary might be able to access. Therefore,  $c\%$  will symbolize the adversary's estimate, and depending on the adversary's strength, the estimate will differ. The objective is to get the absolute number of files that a potential adversary is estimated to access at  $c\%$ . For example, an adversary estimated to access 1% of the user's files is expected to gain access 1 out of 100 files.

### 8.1.2 Metric #2: Differential Privacy

According to Lundmark and Dahlman, the idea behind differential privacy is that “if an adversary cannot differentiate between a data set containing a record, and a data set that does not contain this record the attacker will not be able to infer any additional information” [13]. Furthermore, the formula for differential privacy is [1]:

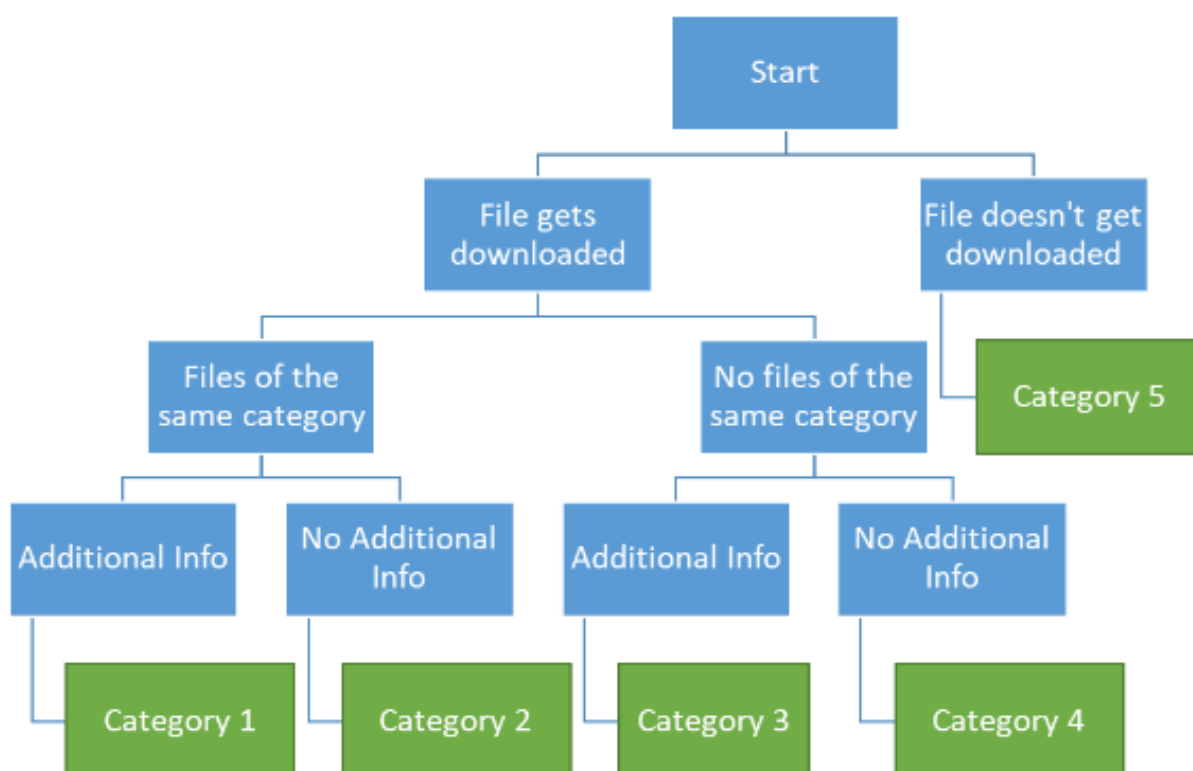
$$\text{privDP} = \forall S \subseteq \text{Range}(K) : p(K(D1) \in S) \leq \exp(\epsilon) \cdot p(K(D2) \in S)$$

where D1 and D2 are data sets that differ in at most a single row. Since this formula is highly complex and does not meet the requirement of being simple, it will be simplified and adjusted to make it applicable for KRAKEN.

The new and less complex metric will also focus on the idea of comparing and differentiating two data sets. As discussed earlier, linkage attacks are a massive threat to people who share information since data might appear anonymized, but in connection to additional data, it could lead to someone's identification. Subsequently, the idea behind KRAKEN's differential privacy is to identify the number of similar files which a user uploaded to the database and find out how likely an adversary is to access at least two similar files.

### 8.1.3 Metric #3: Decision Tree

The objective for this metric is to quantify the scenario where someone downloads a file from KRAKEN, and it utilizes a decision tree to find out how a download affects the user's overall privacy. The decision tree can be seen in Figure 2.



**Figure 2, Decision tree for Metric #3**

Figure 2 shows the different decisions that come into play to determine how a download of the user's files affects their privacy. The tree leads to five categories, where Category 1 has the worst and Category 5 has the best effect on a data subject's privacy. Therefore, depending on the category result,

the user's privacy is affected more or less strongly. The first thing the tree tries to identify is if a file was downloaded or not. If no file got downloaded, the user's privacy is not affected and stays the same. This scenario is called Category 5. If a file got downloaded, then the decision tree would check for further information on the file. First, the tree would need to find out if there exist files of the same category within the user's database. Second, the tree would check if there exists additional information about the file. Additional info is described as available data outside of KRAKEN, which a third person could access. In order to implement this, each category within KRAKEN should have a collection of such data so that the decision tree can see if there is additional info available for a specific category. For example, for GPS-related data, corona heat maps might be identified as additional information.

If there are multiple files of the same category, and there is also additional information available, a download would be associated with Category 1. Subsequently, the user's privacy value will be lowered by  $x$ . But if there exists no additional information, then it will be considered Category 2, and the user's privacy will decrease by  $y$ , where  $y < x$ .

For the case that there exists no other file of the same category, but there is additional information available for the file's category, then its download will be Category 3. The user's privacy will then be decreased by  $z$ , where  $z < y$ . The last scenario quantifies the scenario that there are no additional files of the same category, and there exists no additional information for that category. In this case, the download would be considered a Category 4 download, where the user's privacy decreases by  $w$  and  $w < z$ .

The values for how much the user's privacy is affected after a download can be freely chosen since there is little to no information. Additionally, the category value should not stay constant and change depending on the number of downloads. The more often the same file gets downloaded, the smaller the impact on the user's privacy. For example, the first hundred downloads should more significantly impact the user's privacy than the 1,000,000th download.

Subsequently, this subsection proposes using an exponentially decreasing function that adapts the value of a category with each download of a file. For the simplicity of this Section, the following function is kept short and easy, but it can be changed in any way to fit one's needs better.

$$\text{priv}_{res} = \text{priv} - x^n$$

where  $\text{priv}$  is the current privacy value,  $x$  is the figure that illustrates the category value, and  $x \leq 0$ , and  $n$  is the number of times a certain file has been downloaded.

#### 8.1.4 Utilization

The two-privacy metrics are supposed to measure the likelihood that an adversary of a particular strength manages to access at least two data files with similar content. As discussed in Section 4, an adversary might be able to identify a user by utilizing information that doesn't seem sensible, but with additional information, this data can be used against a user.

For example, if a person uploads a recorded run that shows the GPS-tracking, a potential adversary wouldn't be able to use only this information to identify the user. But if there are multiple work-outs in the database, an adversary could combine those files to conclude where the user might live. Consequently, the suggested privacy metrics aim to notify and show the user their decreasing privacy if they upload multiple files of the same type or files with similar data.

Table 9 shows the necessary conversions to quantify the adversary strengths and privacy values. The range presentation for the adversary strength shows how many files in percent an adversary of the given strength might manage to access. Those ranges can be adjusted and are not an accurate estimate of an adversary's success, and depending on the database, those ranges will vary. The only thing that should be considered is that there must be six intervals to represent each adversary strength so that these ranges align with the privacy values.

Moreover, the privacy values can also be changed depending on how many different privacy values are needed or preferred. For example, one could also use only four privacy values from 1 to 4. Consequently, this table can be adjusted in any way as long as the number of value ranges and privacy values are the same.



Value Range	Category	Privacy Value
100% - 84%	Very Strong Adversary / Very Low Privacy	1
<84% - 68%	Strong Adversary / Low Privacy	2
<68% - 52%	Moderately Strong Adversary / Moderately Low Privacy	3
<52% - 40%	Moderately Weak Adversary / Moderately High Privacy	4
<40% - 24%	Weak Adversary / High Privacy	5
<24% - 0%	Very Weak Adversary / Very High Privacy	6

**Table 9, Grading scale for privacy metric calculation**

Once the table is created and every value is defined, it can be used to quantify the user's privacy. The first metric is slightly related to confidence interval width, as mentioned in Metric #1: Confidence Interval Width. In order to get this metric, the user has to decide which adversary strength they are afraid of. After the user decides on an adversary strength, KRAKEN has to calculate the average amount of files an adversary of a specific strength might access. The more files an adversary can access, the lower is the user's privacy. In order to transform the number of accessible files to a privacy value, one must calculate the percent of files the adversary can access and then compare the percent value to the table.

The second metric builds upon the first calculated metric and is based on differential privacy. Therefore, this metric shows the probability that an adversary manages to access at least two similar files. Subsequently, to calculate this metric, the user must give information regarding the number of similar files. A possible way to get this input will be discussed later. Once the number of similar data is received, it should be possible to calculate the second metric by using stochastics. The formula to get the metric's result looks as follows:

Let  $n$  be the number of similar files,  $k$  be the number of files accessed by an adversary minus the number of similar files the adversary is able to access,  $t$  be the number of total files, and  $a$  be the number of files accessed by an adversary then:

$$\frac{(n \cdot 2 * (t - n) \cdot (k - 2)) + (n \cdot 3 * (t - n) \cdot (k - 3)) + \dots + (n \cdot n * (t - n) \cdot (k - n))}{t \cdot a}$$

The result must be compared to Table 9 to get the privacy value for metric 2. Once both metrics are calculated, the average of both values will be shown to the user as their overall privacy. Further explaining regarding the calculation of the KRAKEN privacy metrics can be seen in Upload Example #1 and Upload Example #2.

The third Metric only comes into play once a file gets downloaded. It works by using a decision tree structure to find out how big the download's impact on the user's privacy is.

## 8.2 Upload Example #1

For the first example, let's assume that the user uploaded their first file and wants to protect their data from a moderately weak adversary:

In this scenario, the calculation of the privacy metric is fairly simple since there is not much to do to receive the final value. The first metric that will be calculated is the one that is based on confidence interval width. Since the confidence interval width is 1, and we assume that the adversary can get 40% to 52% of the user's files, the privacy result for this metric is 4 and can be calculated as follows:



$$\begin{aligned}
 1 - (1 * 0.4) &= 0.4 \\
 1 - (1 * 0.52) &= 0.52 \\
 1 - \frac{0.52 + 0.4}{2} &= 0.46 \\
 0.46 / 1 &= 46\%
 \end{aligned}$$

Then one has to look up the value in Table 9, Grading scale for privacy metric calculation, and get the correct privacy value for the result. In this case, the result would be privacy of 3.

Next, the program must get the privacy metric for Metric 2, which is based on differential privacy. Since there is only one file uploaded to the program in this scenario, the user's privacy for this metric is 6 because there is no other file in the database.

In order to get the user's overall privacy, the program must calculate the average value of privacy metric 1 and 2:

$$\frac{4 + 6}{2} = 5$$

As a result, the user has a privacy value of 5. In other words, this means that they have high privacy, according to Table 4.

### 8.3 Upload Example #2

For the first example, let's assume that the same user uploaded 100 files. Since the user decide that they want their data protected from a moderately weak adversary at their first file upload, they do not have to pick again:

Therefore, the user has 100 files in the database and expects that the user can access 40% to 52% of the files. Consequently, the privacy for the metric based on confidence interval width will be calculated as follows:

$$\begin{aligned}
 1 - (100 * 0.4) &= 40 \\
 1 - (100 * 0.52) &= 52 \\
 100 - \frac{40 + 52}{2} &= 46 \\
 \frac{46}{100} &= 46\%
 \end{aligned}$$

Afterward, one has to plug the value into Table 9, to get the privacy result for this metric. In this scenario, the user has a privacy value of 4.

This time the calculation of the second metric is more complex than in Example #1. For the calculation of the metric based on differential privacy, the program needs a second input that gives an insight on how many similar files compared to the uploaded file are in the database. How to obtain this input value will be discussed in Section 9.

But for this example, let's assume that there are four similar files in the database. The metric aims to calculate the probability that an adversary manages to access at least two similar files out of all the user's files. So, in this scenario, the objective is to figure out the probability that an adversary gets at least two of those four files when they are able to access 46 files. The calculation looks as follows:

$$\frac{(4C2 * 96C44) + (4C3 * 96C43) + (4C4 * 96C42)}{100C46}$$

The result is 0.63, which means there is a 63% probability that a moderately weak adversary is able to get at least two files of similar data out of the dataset. Consequently, the user's privacy for metric 2 is 3.

The overall privacy metric is the average of metric 1 and metric 2. Therefore, the user's privacy is as follows:

$$\frac{4+3}{2} = 3.5$$

The overall privacy is 3.5, and the user has moderately high privacy, according to Table 4.

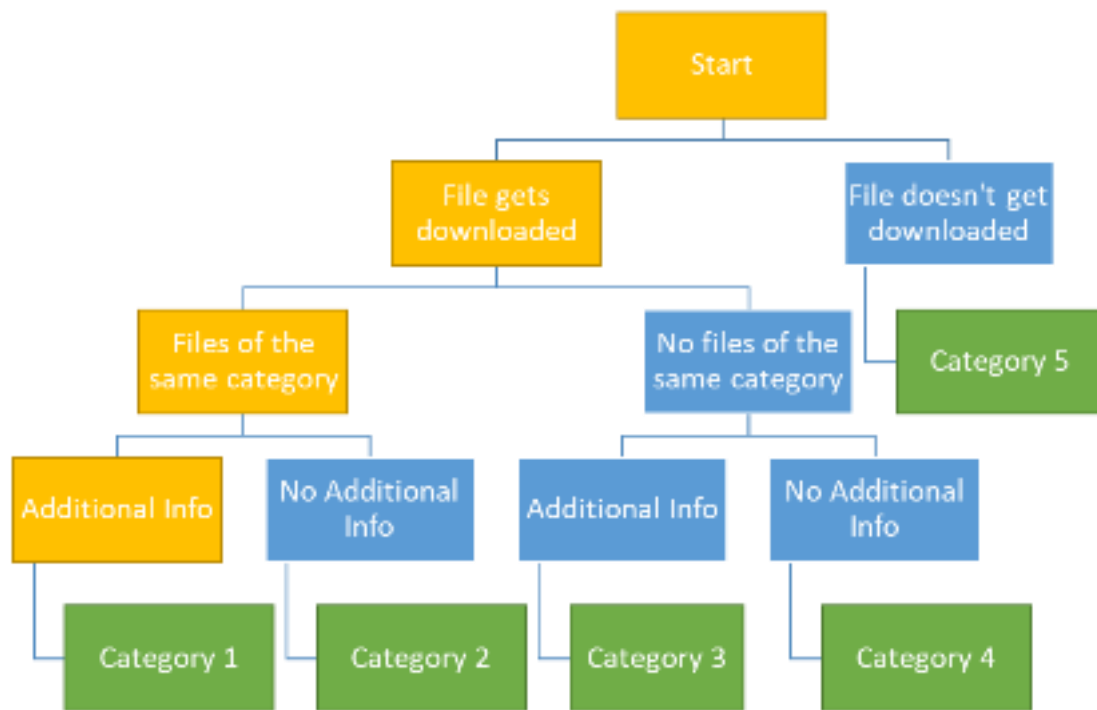
## 8.4 Download Example #1

For illustrating the effects a download might have on a user's privacy, let's build on 8.3. We assume that the user's current privacy is 3.5 or, in other words, the user has a moderately high privacy and that the currently uploaded file gets downloaded. The chosen category values can be seen in Table 10, but it should be mentioned that these values can be adjusted since the ones in the table only have the purpose of showing how this metric works.

Category	Value
Category 1	0.00
Category 2	0.02
Category 3	0.04
Category 4	0.06
Category 5	0.08

**Table 10, Category values**

By utilizing Figure 3, the process can be visualized, and one can see the different steps that happen to figure out the effects the download has on the user's privacy. The yellow fields show the decision tree's path for this example. The first step is that a download takes place. Subsequently, the result must be any of the categories from 1 to 4. Since we already know from 8.3 that the user uploaded files of the same category, the last question that needs to be answered is if additional information outside of KRAKEN is assigned to the file's category type. For this example, we assume that there exist such data. Therefore, the result for this download is Category 1.



**Figure 3, Decision tree example**

From Table 10, we get the value for this category, and then this value must be subtracted from the current privacy value. Consequently, the privacy value after the download is  $3.5 - 0.08^1$ , which is 3.42, and the user has, according to Table 4, moderately low privacy.

## 8.5 Download Example #2

For the second example, let's assume that this is the second download of the same file from 8.4. Therefore, the decision tree will act the same way, and the download is a Category 1 risk for the user's privacy. The only difference from the previous example is that since it is the tenth download, the impact on the user's privacy will be lower. As a result, the privacy value looks as follows:

$$3.42 - 0.08^2 = 3.4136$$

Since, as discussed in Section 8.1.3, this metric utilizes an exponentially decreasing function to get the impact of the download, the privacy value is now less affected than it was the first time. Thus, after the second download, the current privacy value is 3.4136, which means that the user still has moderately low privacy.

## 9. Implementation Concept

The privacy metric's implementation is based on the subsections Input Data, Target Audience, and Quantification. Subsequently, this subsection will be a summary of these three parts. Additionally, one should always keep in mind that the metrics should be implemented to be easy to use and understand.

### 9.1 Input Data

The necessary input data for calculating the privacy metrics are partly dependent on user input and partially reliant on the information coming from the database itself. Once they add their first data files to the database, they must enter the first input variables: adversary strength and file similarity. The

user should have the option to pick between six different adversary strengths from which they want their data protected. We decided on six adversary strengths because there are also six different privacy categories, and to compare those two scales better, they should be of the same size.

For obtaining the user input, it is recommended to have the user fill out a form while uploading a new file. The form could look as follows:

## File Upload on KRAKEN

- Question #1 (Only asked for first upload):

- Adversary Strength?

Very Weak	Weak	Mod. Weak	Mod. Strong	Strong	Very Strong
1	2	3	4	5	6

- Question #2 (always asked):

- Does this file have similar information or does it have relating data to previously uploaded files?

Yes	No
-----	----

- If yes, how many files approximately share similar data? (Files categorizieren)

Enter number:	<input type="text"/>
---------------	----------------------

**Figure 4, User input form 1**

Question 1 only gets asked the first time a user uploads a file, and question 2 gets asked every time a user uploads data to KRAKEN. An additional option for getting the approximate amount of files with similar information would be to categorize them. For example, the user could give information about the type of category the uploaded file belongs to, such as education, health, finances, etc.

The second input comes directly from KRAKEN and describes of how many data files a user uploaded to the database. This one should be relatively simple to implement since this variable only keeps a count of the user uploads. For example, if a user uploads a file, the variable should increase by one, and if they delete a file, the variable should decrease by one.

As shown in Section 8, those inputs should be enough to calculate the privacy metrics and inform the user of their privacy estimate.

## 9.2 Metrics

The quantification of the privacy metrics shouldn't be challenging to implement since most of the inputs come directly from the user, as can be seen in Section 9.1. One variable is the adversary strength, and the other is the number of similar files. Additionally, there is also a third metric that focuses on downloads. Consequently, all KRAKEN has to do is keep track of the number of user files in the database and how often a particular file got downloaded.

The calculation of the metrics itself would be fairly easy since it mainly depends on the adversary strength and only calculates the average amount of files an adversary of a certain strength might be able access. Therefore, the calculation only consist of simple addition, subtraction, multiplication, and division, and might look as follows:

```
def metric1_calculator():
```

```
    adv_strength_min = Lower adversary strength
```

```
    adv_strength_max = Higher adversary strength
```

```
    num_files = Total number of user files
```

```

sim_files = Number of similar files
val1 = num_files – (num_files * adv_strength_min)
val2 = num_files – (num_files * adv_strength_max)
file_access = num_files – ((val1 + val2)/2)
privacy_val = (file_access / num_files) * 100

```

Lastly, the program would need to use `privacy_val` as a key to get the correct privacy estimate from the table, which could be implemented as a hash map, dictionary, etc.

The quantification of the second metric is based on the input `sim_files`, `num_files`, and `privacy_val` and is more complicated. The variable `privacy_val` should be used to get the number of files an adversary might be able to get and then this new variable should, together with the other two variables, calculate the probability that the adversary gets at least two of `sim_files` when being able to access `num_files * (privacy_val/100)`. The pseudo-code for this part might look as follows:

```

def metric2_calculator():
    file_access = num_files * (privacy_val/100)
    total = 0
    temp_files = num_files – sim_files
    for i in range(2, sim_files + 1):
        temp_access = file_access - i
        total += ncr_combination(sim_files, i) * ncr_combination(temp_files,
            temp_access)
    result = total / (ncr_combination(num_files, file_access))
    result = result * 100
    return result

def ncr_combination(n, r):
    result = helper_comb(n) / ((helper_comb(r))*(helper_comb((n – r))))

def helper_comb(val):
    if val == 1:
        return 1
    return val * helper_comb(val-1)

```

The third metric can be easily implemented by using conditional statements. All it needs is conditional statements that check, if a file gets downloaded, if there exist files of the same category, and if there is additional information available. Depending on the outcomes of these statements, the according category value will be subtracted from the privacy value. For example,

```

def metric3():
    if file gets downloaded:
        if there are files of the same category:
            if there exists additional info:
                return Category_5
            else:
                return Category_4
        else:
            if there exists additional info:

```

```

        return Category_3
    else:
        return Category_3

    else:
        return Category_1

def metric3_calculator(file):
    value = metric_3(file) ** download_count
    privacy_val -= value
    return privacy_val

```

### 9.3 Output Data

As discussed previously, KRAKEN's goal is to implement the privacy metrics so that they are easy to understand and don't create difficulties for the users. Furthermore, the user should be able to understand the output of their privacy estimate regardless of their technical expertise. As a result, in part 7, Target Audience, we mentioned that the output could be portrayed in a similar way to most password strengths output. An example of such an output can be seen in Figure 1 or Figure 5.

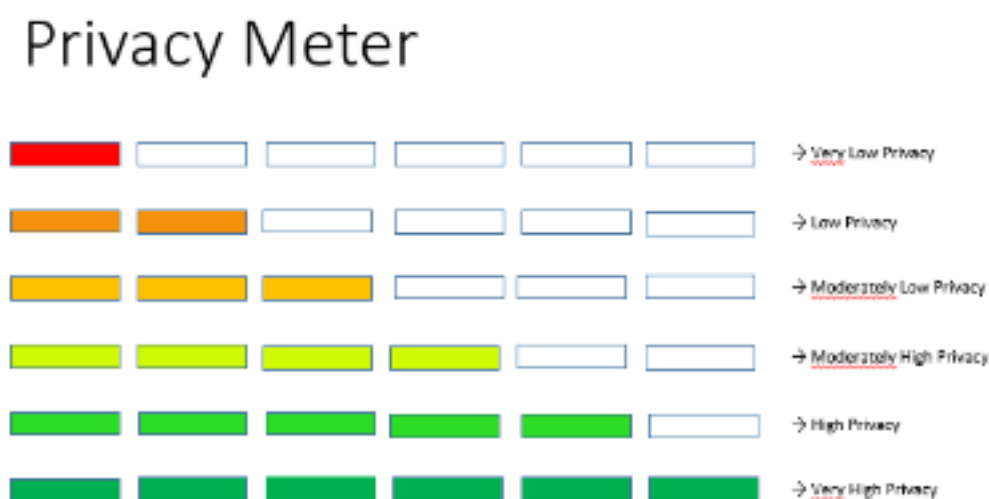


Figure 5, Privacy meter

Besides, the user would benefit from receiving information on why their privacy estimate is classified as it is and how they could improve their privacy. Subsequently, it should be considered to inform the user about their similar files and tell them that they should consider deleting some of the similar files to improve their privacy if possible.

## 10. Conclusion

This section presented ideas that should help to quantify the users' privacy within the KRAKEN database. Most of the recommendations can be easily adjusted and changed by using different metrics or weighing the importance of certain metrics more than others. The chosen privacy metrics defined in this section are based on confidence interval width, differential privacy, and decision trees and aim to give the user insights on their privacy by focusing mainly on linkage attacks. All metrics are kept as simple as possible to facilitate their implementation and make them easier to understand. Therefore, they were strongly modified to fit better into the KRAKEN context.

Furthermore, this section offers the framework to choose other privacy metrics, if needed. Depending on what one tries to quantify, different metrics apply. This part also showed how privacy metrics could be added to already implemented metrics. That should allow for updating the KRAKEN privacy metrics regularly, if necessary, and make the existing system even more compact to give the user an even better quantification of their privacy.

## 11. Bibliography

- [1] I. Wagner and D. Eckhoff, “Technical privacy metrics: a systematic survey,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–38, 2018.
- [2] Koch, K., Krenn, S., Pellegrino, D., and Ramacher, S. „Privacy-Preserving Analytics for Data Markets Using MPC“, o. J., 20.
- [3] KRAKEN Consortium, “KRAKEN\_D2.6 Marketplace Technical Specification\_v1.0”, pp. 1-49, 2020.
- [4] Privitar. Linkage Attack. Retrieved (2021) from <https://www.privitar.com/glossary/linkage-attack/>
- [5] J. D. Cook Consulting. “Simulating identification by zip code, gender, birthdate”. 2018. Retrieved from <https://www.johndcook.com/blog/2018/12/07/simulating-zipcode-sex-birthdate/>
- [6] Wired. “Why ‘Anonymous’ Data Sometimes Isn’t”. 2007. Retrieved from <https://www.wired.com/2007/12/why-anonymous-data-sometimes-isnt/>
- [7] Narayanan, A., & Shmatikov, V. “How To Break Anonymity of the Netflix Prize Dataset. Arxiv cs/0610105”. 2006. Retrieved from <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.3581&rep=rep1&type=pdf>
- [8] Barth-Jones, D. C. The „Re-Identification of Governor William Weld’s Medical Information: A Critical Re-Examination of Health Data Identification Risks and Privacy Protections, Then and Now”. SSRN Electronic Journal. 2012. Retrieved from <https://doi.org/10.2139/ssrn.2076397>
- [9] Ohm, P. “BROKEN PROMISES OF PRIVACY: RESPONDING TO THE SURPRISING FAILURE OF ANONYMIZATION”. UCLA LAW REVIEW, 77. 2012. Retrieved from <http://www.uclalawreview.org/pdf/57-6-3.pdf>
- [10] “Joint Task Force Transformation Initiative. “Guide for conducting risk assessments (NIST SP 800-30r1; 0 Aufl., S. NIST SP 800-30r1)”. National Institute of Standards and Technology. 2012. Retrieved from <https://doi.org/10.6028/NIST.SP.800-30r1>
- [11] Samarati, P., & Sweeney, L. (o. J.). “Generalizing Data to Provide Anonymity when Disclosing Information”. 15. Retrieved from <https://dataprivacylab.org/dataprivacy/projects/kanonymity/paper4.pdf>
- [12] Lundmark, M., & Dahlman C. “Differential privacy and machine learning: Calculating sensitivity with generated data sets.” KTH Royal Institute of Technology. 2017. Retrieved from <https://kth.diva-portal.org/smash/get/diva2:1112478/FULLTEXT01.pdf>
- [13] Cynthia Dwork. 2006. Differential Privacy. In Proc. 33rd Int. Colloq. on Automata, Languages and Programming (ICALP2006) (LNCS 4052). Retrieved from <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/dwork.pdf>





**Atos**

**F3K**  
FONDAZIONE  
BRUNO KESSLER

**AIT**  
AUSTRIAN INSTITUTE  
OF TECHNOLOGY



**LYNKEUS.**  
STRATEGY CONSULTING | BLOCKCHAIN & SMART CONTRACTS | DATA ANALYTICS



**TX**

**KU LEUVEN** **CITIP**  
CENTRE FOR IT & IP LAW

**IAIK** **TU**  
Graz

**InfoCert**  
TINEXTA GROUP

@KrakenH2020



Kraken H2020



[www.krakenh2020.eu](http://www.krakenh2020.eu)



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