D3.3 Smart Contracts

Authors: **Steffen Biehs, Jan Jürjens (FhG), Alan Barnett, Matthew Keating (EMC), Lorenzo Gugliotta, Yuliya Miadzvetskaya, Lidia Dutkiewicz, Charlotte Ducuing (KUL)**

December 2022

TRUSTS Trusted Secure Data Sharing Space

D3.3 Smart Contracts

Document Summary Information

Grant Agreement No	871481	Acronym	TRUSTS
Full Title	TRUSTS Trusted Secure Data Sharing Space		
Start Date	01/01/2020	Duration	36 months
Project URL	https://trusts-data.eu/		
Deliverable	D3.3 Smart Contracts		
Work Package	WP3 Platform		
Contractual due date	31/12/2022	Actual submission date	21/12/2022
Nature	Report	Dissemination Level	Public
Lead Beneficiary	FhG		
Responsible Author	Jan Jürjens (FhG), Steffen Biehs (FhG)		
Contributions from	Alan Barnett (EMC), Lorenzo Gugliotta (KUL), Yuliya Miadzvetskaya (KUL), Lidia Dutkiewicz (KUL), Charlotte Ducuing (KUL)		

Revision history (including peer reviewing & quality control)

Version	Issue Date	% Complete	Changes	Contributor(s)
v1.0	26/03/2020	0	Initial Deliverable Structure	Bin Li (FhG)
v1.1	03/09/2021	20	Input from partners	FhG, Dell, KUL
v1.2	29/10/2021	35	Input in background, new chapters	FhG
v1.3	28/02/2022	75	Extend chapter 4, new chapter 7 about the smart contract demonstrator	KUL, Dell
v1.4	24/05/2022	85	Extension of chapter 4, update other chapters	KUL, Dell, FhG
v1.5	15/07/2022	90	Finish chapter 3	FhG
v1.6	25/08/2022	95	Finish chapter 5	FhG
V1.7	15/11/2022	98	Finish chapters	FhG, Dell
V1.8	17/11/2022	99	Add Introduction and Conclusion	FhG
V1.9	20/12/2022	100	Add review from partners	KUL, Dell, FhG

Disclaimer

Copyright message

Table of Contents

1 Executive Summary 10
2 Introduction 11
2.1 Mapping Projects' Outputs 11
2.2 Deliverable Overview and Report Structure 12
3 Background 14
3.1 Data Markets 14
3.1.1 General Information 14
3.1.1.1 Actors and Roles 15
3.1.1.2 Types and Domains of Data Marketplaces 15
3.1.2 Data Market Austria 16
3.1.3 International Data Spaces 17
3.1.3.1 Goals of the International Data Spaces 17
3.1.3.2 Data exchange and data sharing 17
3.1.3.3 Roles in the International Data Spaces 18
3.1.3.4 Core Participants 18
3.1.3.5 Intermediary 19
3.1.3.6 Software / Service Provider 19
3.1.3.7 Governance Body 19
3.1.3.8 The International Data Spaces Association 20
3.1.4 TRUSTS Architecture 20
3.2 Blockchain 21
3.3 Smart Contracts 23
4 Legal Challenges 25
4.1 Introduction to Smart Contracts 25
4.2 Applying Contract Law to Smart Contracts: Legal Challenges 26
4.2.1 Legal qualification and validity of smart contracts 26
4.2.1.1 Legal qualification of smart contracts 27
4.2.1.2 Validity of smart contracts 30
4.2.2 Enforceability 32
4.2.2.1 Consequences and challenges related to immutability 32
4.2.2.2 Remedial action 34
4.2.3 Language and interpretation 34
4.2.3.1 The challenge of translating human legal language into code 35

4.2.3.2 Challenges and implications stemming from coding errors or ambiguities 36
4.2.4 Cybersecurity risks 37
5 Smart Contracts in TRUSTS 39
5.1 Data Transactions and Smart Contracts in TRUSTS 39
5.1.1 Requirements 40
5.1.1.1 Functional Requirements 40
5.1.1.2 Technical Requirements 41
5.1.2 Technical Implementation 42
5.1.3 Use in a TRUSTS data market 43
5.2 Complications of Smart Contracts in TRUSTS 45
6 Smart Contract Privacy and Security 48
6.1 Privacy of Smart Contracts 48
6.1.1 Privacy Preserving Smart Contracts 48
6.1.1.1 Private and Consortium Blockchain Variants 48
6.1.2 Smart Contract Privacy Vs Confidentiality 48
6.1.3 Overview of Privacy Preserving Technologies for Smart Contracts 49
6.1.3.1 Encrypted On-Chain Data with Homomorphic Encryption 49
6.1.3.2 Secure Multi-Party Computation 49
6.1.3.3 Secure Enclave 49
6.1.3.4 Zero Knowledge Proofs 50
6.2 Security of Smart Contracts 50
6.2.1 Smart Contract Security Principles 50
6.2.2 Common Smart Contract Vulnerabilities 51
6.2.2.1 Denial of Service Attacks 52
6.2.2.2 Overflows - Integer and Numeric 53
6.2.2.3 Storage Injection 53
6.2.2.4 Remote Code Execution 53
6.2.2.5 RAM Exploit 53
6.2.2.6 Re-Entrancy Attack 54
6.2.2.7 Updating On-chain Smart Contracts 54
6.3 Integrity of Smart Contracts 54
7 Smart Contract Demonstrator 56
7.1 Blockchain - Hyperledger Fabric Software 56
7.2 Hyperledger Explorer UI 57
7.2.1 Explorer UI Dashboard Overview 57

7.2.2 Explorer UI Search 58
7.3 Contract Library 58
7.3.1 Basic Asset Transfer 59
7.3.2 Secured Asset Transfer 59
7.3.3 Review Score 60
7.3.4 Query Blockchain 60
7.3.5 Properties List for Smart Contract Demonstrator 61
7.4 NodeJS API 62
7.4.1 User Accounts & Access Control 62
7.4.2 Fabric Gateway 63
7.4.3 Chaincode Execution 63
7.4.4 Blockchain Search 64
7.4.5 Transaction ID Return and Explorer UI Link 64
7.4.6 Demonstrator Client 65
7.5 Payment Compatibility Demonstrator 65
8 Conclusions and Next Actions 69
9 References 70

List of Figures

Figure 1: Categorization of electronic data markets based on their provider/consumer relations [1]. 16
Figure 2: Roles and Interaction in the Industrial Data Space. 18
Figure 3: TRUSTS Architecture. 21
Figure 4: Illustration of a blockchain. The main chain (black) consists of the longest series of blocks from the genesis block (green) to the current block. Orphan blocks (purple) exist outside the main chain. 22
Figure 5: Smart Contract Lifecycle. 39
Figure 6: Adding a new block to the blockchain while an asset shall be transferred. 42
Figure 7: Trigger a smart contract 44
Figure 8: Smart contract demonstrator architecture. 56
Figure 9: Hyperledger Explorer UI dashboard & sample data. 58
Figure 10: Basic asset transfer blockchain properties with sample values. 61
Figure 11: Secured asset transfer blockchain properties with sample values. 62
Figure 12: User scoring blockchain properties with sample values. 62
Figure 13: Hyperledger Explorer UI representation of blockchain entry. 65
Figure 14: Web UI of payment system compatibility demonstrator. 66
Figure 15: Out-Of-Band payment system order page example. 67
Figure 16: Invoice example. 68

List of Tables

Table 1: Technical Requirements. 41
Table 2: Chaincode types. 43
Table 3: Risks and drawbacks using smart contracts. 45
Table 4: Defects using smart contracts. 46
Table 5: Smart Contract attacks on Blockchain Networks [37, 38, 39, 40] 52

Glossary of terms and abbreviations used

Abbreviation / Term	Description
IDS	International Data Spaces
IDSA	International Data Spaces Association
HLF	Hyperledger Fabric
CKAN	Comprehensive Knowledge Archive Network
DMA	Data Market Austria
BMVIT	Austrian Ministry of Transport, Innovation and Technology
DLT	Distributed Ledger Technology
DSA	Digital Services Act
DCFR	Draft Common Frame of Reference
API	Application Programming Interface
SGX	Software Guard Extensions
TEE	Trusted Execution Environment
sMPC	Secure Multi-Party Computation
EOS	Crypto Coin
NEO	Crypto Coin
SSH	Secure Shell
RAM	Random-Access Memory
REST	Representational State Transfer
SDK	Software Development Kit
UI	User Interface
ID	Identifier
gRPC	gRPC Remote Procedure Calls
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
GPL	GNU General Public License
UUID	Universally unique identifier

1 Executive Summary

This deliverable is part of the Work Package 3 “Platform” of the “TRUSTS - Trusted Secure Data Sharing Space” project and summarizes the efforts taken with regards to Task 3.3 “Smart Contracts”.

The subject of this deliverable is to develop the necessary concepts for using smart contracts in the context of the European Data Market TRUSTS. The approach to achieve this was to conduct a literature review, to evaluate the requirements collected by the project partners which are relevant for Task 3.2, and to consider the project goals of TRUSTS in order to develop these necessary concepts. As a result, various challenges could be derived for both the technical and the legal perspective on the topic, which are examined in the respective chapters.

During the development of the concepts, it became clear that for TRUSTS, the legal aspects in particular play a decisive role. The approach of a European data market brings together different legal forms. Therefore, a common consensus must be found there for the use of smart contracts. When working out the technical aspects, the focus was primarily on the core idea of TRUSTS: security and trustworthiness. Therefore, potential sources of error for the technical use of smart contracts were elaborated and explained, and recommendations were made as to which technologies with their characteristics are best suited for a TRUSTS data market. From the resulting recommendations, a smart contract demonstrator was developed, which takes these into account and thus represents a minimal example for the implementation of a blockchain with smart contract execution in TRUSTS.

2 Introduction

This deliverable reports the status of Task 3.2 “Smart Contracts”. The main goal of the task is to develop the necessary concepts for using smart contracts in the context of the European data market delivered by TRUSTS. The goal focuses on a technical and a legal perspective. The technical objectives of this TRUSTS deliverable are the foundations for ensuring the integrity and authenticity of smart contracts as well as the challenges related to the fact that smart contracts are written in executable code which is supposed to be a self-sufficient representation of a natural language contract which will be studied. The legal objectives of this TRUSTS deliverable are to analyze the legal challenges brought about smart contracts, such as issues of validity, enforceability, and interpretation as well as the legal issues and limitations related to the fact that smart contracts are written in code instead of natural language will be studied.

The approach to achieve the goal of Task 3.2 was to conduct a literature review, to evaluate the requirements collected by the project partners which are relevant to Task 3.2, and to consider the project goals of TRUSTS in order to develop a deliverable with the necessary concepts for using smart contracts in the context of the European data market delivered by TRUSTS.

The objectives of Task 3.2 are aligned with the objectives of the overall project in that TRUSTS aims to provide various components for a secure European data market and Task 3.2 describes the concepts for one of these components. A data market lives from transactions among the participants. However, in order to be able to clarify problems and disputes quickly, a component is needed within the data market that can provide a legally and technically secure overview of all transactions in the market. In this component, all transactions are stored in a blockchain that were created between the participants in the past. Manipulation can thus be prevented, which in turn increases security in the data market and trust between the participants.

The results of this task could be leveraged in a number of ways, such as using the privacy and security findings to develop a smart contract system to defend against some of the most high-profile attacks on these types of systems, in addition to protecting European data markets along the lines of TRUSTS. Dell EMC intends to use the results as a basis for further investigation of blockchain and smart contract security issues after the project. It also aims to demonstrate beyond that how to use more conventional payment systems in conjunction with smart contracts without sacrificing blockchain integrity guarantees, as opposed to using tokens or cryptocurrencies in conjunction with blockchain. In both security research and payment systems research, the work of Task 3.2 will continue after the project ends.

2.1 Mapping Projects’ Outputs

Purpose of this section is to map TRUSTS Grand Agreement commitments, both within the formal Deliverable and Task description, against the project’s respective outputs and work performed.

Table 1: Adherence to TRUSTS GA Deliverable & Tasks Descriptions

TRUSTS Task		Respective Document Chapter(s)	Justification
T3.2 Smart Contracts	FhG-ISST will develop the necessary concepts for using smart contracts in the context of the European data market delivered by TRUSTS. In particular, this includes the technical foundations for ensuring the integrity and authenticity of such contracts. The technical challenges related to the fact that smart contracts are written in executable code which is supposed to be a self-sufficient representation of a natural language contract will be studied. KUL will analyze the legal challenges brought about smart contracts, such as issues of validity, enforceability, and interpretation. In particular, the legal issues and limitations related to the fact that smart contracts are written in code instead of natural language will be studied. Collaboration between the legal and technical partners involved in the coding of smart contracts will ensure that these contracts include necessary legal and technical information while respecting the privacy by design and privacy by default principles. The findings of this task will be included in deliverable D3.3.	Chapter 4 Chapter 5 Chapter 6 Chapter 7	*These chapters describe the whole development process of concepts for using smart contracts in the context of the European data market delivered by TRUSTS. Starting with the legal challenges, over the technical challenges and privacy and security aspects, to a demonstrator for such a blockchain component with smart contract execution.*
TRUSTS Deliverable			
D3.3 Smart Contracts This deliverable will provide the results from T3.2, namely the necessary concepts for making use of smart contracts at the technical level within the TRUSTS European Data Market and the overview of relevant legal frameworks and respective legal challenges.			

2.2 Deliverable Overview and Report Structure

The document is part of Task T3.2 "Smart Contracts", in which the necessary concepts for using smart contracts in the context of the European data market delivered by TRUSTS will be developed. The concepts were considered from both a technical and legal perspective to provide a holistic view of the issues and challenges in implementing and using smart contracts in the European data market TRUSTS. The deliverable is therefore divided into different chapters, which are described in more detail below:

- **Section 3 “Background”** Chapter 3, "Background," provides an overview of the fundamentals of the topic. It begins by describing what a data market is and goes into more detail on the two data markets that shape TRUSTS, DMA and IDS. This is followed by definitions and explanations of blockchains and smart contracts and how each works. Finally, a few examples of smart contract applications in data markets are listed.
- **Section 4 “Legal Challenges”** describes, in addition to a general legal overview of smart contracts, the legal challenges of applying contract law to smart contracts and how the whole subject can be applied in compliance with EU law.
- **Section 5 “Smart Contracts in TRUSTS”** describes the technical challenges of implementing smart contracts in the European data market TRUSTS. In addition to the functional and technical requirements for a blockchain application with smart contracts execution, an explanation is given of how these requirements can be met. Furthermore, there will be an illustration of how such a blockchain application can be connected in TRUSTS using the example of a sequence diagram. Finally, an overview of various problems, risks, and drawbacks in the use of blockchains and smart contracts is presented, which must be taken into account in the technical implementation of a blockchain application with smart contracts execution.
- **Section 6 “Smart Contract Privacy and Security”** provides an overview of methods and potential problems that arise in the privacy and security of smart contracts. It explains various techniques for maintaining a high level of privacy and security, as well as highlighting vulnerabilities, explaining their causes, and showing how they can be addressed.
- **Section 7 “Smart Contract Demonstrator”** describes a smart contract demonstrator based on all the previously mentioned sections, which, in addition to a pure blockchain with smart contract execution, has a UI, a library of various smart contracts and its own API. The latter is also used to demonstrate payment compatibility, which means that the smart contract demonstrator can theoretically also be used for payment transactions.

3 Background

In this chapter, background descriptions of all T3.2 relevant topics are given.

3.1 Data Markets

In this section, there will be an overview of different data markets frameworks, namely Data Market Austria and the International Data Spaces. The words data market and data marketplace will be used synonymous.

3.1.1 General Information

The information in this subchapter is divided into three parts: A general definition about data markets, the different roles within a data market, and data market types.

A data marketplace is a multi-sided trading platform where participants can share/sell (producers) and buy (consumers) datasets.

Access to the data, manipulation, and the use of the data by consuming entities is commonly governed by the data marketplace using a range of standardized or negotiated licensing models. The marketplace also mediates the interactions between various actors in the ecosystem. Data marketplace platforms require specific governance mechanisms via legal protection, quality control and comprehensive data provenance. [3] Thereby, data marketplaces will foster innovative business models where data is the raw material – data becomes the primary commodity. [4]

In principle, data marketplaces could resemble multisided platforms, where a digital intermediary connects data providers, data purchasers, and other complementary technology providers. Such platforms could generate value for both data buyers and sellers through lower transactional frictions, resource allocation efficiency, and improved matching between supply and demand. [5]

Data marketplaces are electronic platforms that facilitate the exchange of data. Due to the need for businesses to obtain appropriate information and the maturation of the data market ecosystem, the popularity of data marketplaces has grown in recent years. As the number of data marketplace players joining the market increases, companies have more opportunities to use external data to improve their business and explore new revenue opportunities by reselling the data they collected internally.

A data marketplace ecosystem consists of data providers, data buyers, third-party service providers and a marketplace owner. Data providers offer their own data on a marketplace; allow their data to be queried by data buyers and expect to obtain revenue by selling data. Data buyers are participants who are interested in buying the datasets they need and display a positive willingness to pay for data. Data buyers use purchased data to support decision-making processes or build new services and business models. Third-party service providers can provide applications or algorithms that add value to data assets. The data marketplace provider collects and hosts data from data providers and sells data to data buyers.

According to, an electronic marketplace is a data marketplace if data trading is the main value proposition of the business model. Marketplace participants must be able to upload, browse, download, buy, and/or sell machine-readable data. Consequently, this excludes the services that only offer links to data locations,

without hosting the data. Moreover, data must be hosted by the providers who clarify the origin of the data. Additionally, data marketplaces must be digital platforms and not only a data repository or a cloud service provider. [6]

3.1.1.1 Actors and Roles

Participants in a data marketplace take on different roles, such as data provider and data consumer. The data marketplace itself also provides actors to provide various services within the market. These include, for example, the data broker and the infrastructure provider. The owner of the data marketplace also represents a role on its own. The following is an overview of roles in a data marketplace:

- **Data Providers**: Provide and offer their data, allow their data to be queried, obtain revenue by selling data
- **Data Consumers**: Interested in buying the datasets they need, utilize the data available in the data marketplace platforms directly
- **Service Providers**: Offer data-driven services such as visualizations and analytics. This can also be a third-party participant.
- **Infrastructure and Tool Providers**: Deliver the required technical components and tools to make the ecosystem works
- **Data Brokers**: Establish a link between data providers and other actors in the ecosystem
- **Application Users**: Make use of the data-driven applications and services to consume the data
- **Marketplace Owners**: Collects and hosts data from data providers and sells data to data consumers

Data consumers and application users are almost the same but differ in the fact that applications users are real users and data consumers can also be a company or other services who consume data. [3, 4, 5, 6]

3.1.1.2 Types and Domains of Data Marketplaces

Data markets exist in a wide variety of flavors and for different domains. Definitions of data markets are diverse and heterogeneous. Stahl et al. [1] divide electronic data markets into different categories based on their provider/consumer relations. Data markets can serve single or multiple consumers and single or multiple providers (see Figure 1). In some cases, the single provider is equivalent to the data market operator itself. Data markets are categorized in three different sectors: private, consortium and independent data markets. A private data market has either a single buyer or single supplier, i.e., it offers one-to-many or many-to-one relations to the sides of buyers/suppliers. Consortium data markets are unions of either suppliers and buyers, or both. The third type, the independent data market, serves as an arbiter for buyers and suppliers, i.e., providing a platform and infrastructure for successful search, acquisition, accounting, and transfer of data. They allow for many-to-many relations between independent players in the data market.

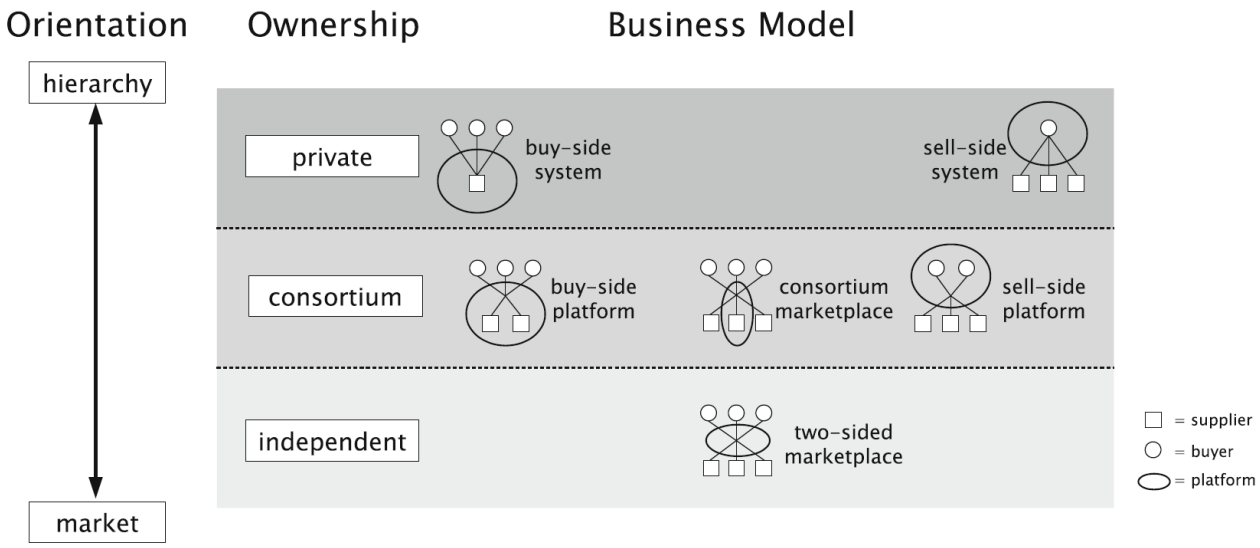

Figure 1: Categorization of electronic data markets based on their provider/consumer relations [1].

In contrast to this lenient categorization of Data markets, the IDS Reference Architecture Model [2] provides a much stricter definition. Data markets are exclusively limited to the independent type, i.e. require the conceptual availability of m:n relations between independent buyers and suppliers. Data providers such as Facebook, who sell their data products to external organizations, are consequently not considered as a Data market. Alternatively, the definition of Stahl et al. [1] would allow those to be considered as a Data market.

Data markets cover a wide range of different domains, such as agriculture, finances, healthcare, location services, etc. According to [2], the by far most prominent domain is the so-called audience data, providing data from the marketing and advertising industry.

3.1.2 Data Market Austria

In 2016, the Data Market Austria (DMA) project started with a consortium of 17 Austrian partners. The DMA is partially funded by the Austrian Ministry of Transport, Innovation and Technology (BMVIT) and the Austria Research Promotion Agency (FFG). The DMA project aims to provide a data innovation environment by improving technology for secure data marketplaces and cloud interoperability. The ecosystem is supporting a full spectrum of data, i.e., both open and business data, with a sufficient level of access control [7].

The DMA does not host the data, but it provides a catalogue for registered datasets and facilitates data exchange between core actors (data providers and data customers). The DMA developed technological innovations for data exchange, such as blockchain mechanisms to ensure data provenance and security, recommender systems based on brokerage technology and semi-automated data quality improvement [7].

The target group of DMA customers comprises a broad range of data-driven organizations and individuals, including non-profit & civic society, scientific & research, public sector government & admin, and private sector [8].

The DMA was an Austrian key project to conceptualize, implement, and establish an Austrian ecosystem for data services. From a technical perspective, the DMA was a federated platform leveraging a microservices architecture, data and metadata harvester and ingestion services, a recommendation engine, and distributed ledger technology for smart contracting.

The DMA accomplished a lot of basic research work to elicit and specify the requirements of a data market including its objectives, its role in an ecosystem of demand and supply regarding data trading and exchange, as well as in specification of relevant roles and stakeholders for a data market and in respect to components, features and technologies, as well as the overall architecture of a data market. Findings are well documented and publicly available and, several software components are available as open-source software.

3.1.3 International Data Spaces

The International Data Spaces (IDS) are a virtual data space leveraging existing standards and technologies, as well as governance models well-accepted in the data economy, to facilitate secure and standardized data exchange and data linkage in a trusted business ecosystem. It thereby provides a basis for creating smart-service scenarios and facilitating innovative cross-company business processes, while at the same time guaranteeing data sovereignty for data owners.

Unless otherwise indicated, this section refers to IDS reference architecture v.3.0 [2], as discussed by Pettenpohl et al. [85].

3.1.3.1 Goals of the International Data Spaces

The main goal of the IDS is data sovereignty. Data sovereignty is a central aspect of the International Data Spaces. It can be defined as a natural person's or corporate entity's capability of being entirely self-determined with regard to its data. The Reference Architecture Model particularly addresses this capability, as it specifies requirements for secure data exchange and restricted data use in a trusted business ecosystem. The International Data Spaces promotes interoperability between all participants based on the premise that full self-determination with regard to one's data goods is crucial in such a business ecosystem. Data exchange takes place by means of secured and encrypted data transfer including authorization and authentication. The Data Provider may attach metadata to the data transferred using the IDS Vocabulary. In doing so, the terms and conditions to ensure data sovereignty can be defined unambiguously (e.g., data usage, pricing information, payment entitlement, or time of validity). The International Data Spaces thereby supports the concrete implementation of applicable law, without predefining conditions from a business point of view, by providing a technical framework that can be customized to the needs of individual participants.

3.1.3.2 Data exchange and data sharing

Cross-company data exchange with the help of inter-organizational information systems is not a new topic; it has been around for decades. With the proliferation of Electronic Data Interchange (EDI) in the 1980s, many different data exchange scenarios have emerged over time, which were accompanied by the development of certain technical standards.

Data sovereignty, which is one of the main goals of the International Data Spaces, materializes in "terms and conditions" that are linked to data before it is exchanged and shared. However, these terms and conditions (such as time to live, forwarding rights, pricing information etc.) have not been standardized

yet. In order to foster the establishment of data sovereignty in the exchange of data within business ecosystems, more standardization activities are needed.

This does not mean that existing standards will become obsolete. Instead, the overall set of standards companies need to comply with when exchanging and sharing data needs to be extended. It is therefore necessary to distinguish between data exchange and data sharing:

- Data exchange takes place in the vertical cooperation between companies to support, enable or optimize value chains and supply chains.
- Data sharing takes place in the vertical and horizontal collaboration between companies to achieve a common goal (e.g., predictive maintenance scenarios in manufacturing) or to enable new business models by generating additional value out of data (e.g., in data marketplaces). Furthermore, data sharing implies a mode of collaboration towards coopetition.

3.1.3.3 Roles in the International Data Spaces

In the following, each role a participant can assume in the International Data Spaces is described in detail, together with the basic tasks assigned to it. The majority of roles require certification of the organization that wants to assume that role, including certification of the technical, physical, and organizational security mechanisms the organization employs. Certification of organizations that want to participate in the International Data Spaces is considered a fundamental measure to establish trust among all participants (especially with regard to roles that are crucial for the overall functioning of the International Data Spaces, such as the Broker Service Provider, the App Store, the Identity Provider, or the Clearing House). An overview over all roles in die IDS is given in Figure 2.

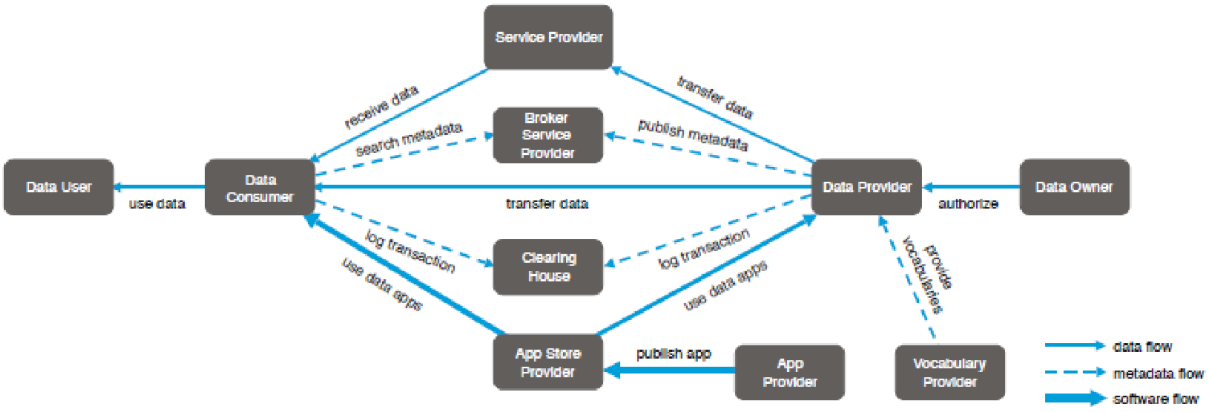

Figure 2: Roles and Interaction in the Industrial Data Space.

3.1.3.4 Core Participants

Core Participants are roles involved and required every time data is exchanged in the International Data Spaces. Roles assigned to this category are Data Owner, Data Provider, Data Consumer, Data User, and App Provider. The role of a Core Participant can be assumed by any organization that owns, wants to provide, and/or wants to consume or use data. Benefits for participants in the International Data Spaces are created by these roles as they make data available (Data Owner), provide data (Data Provider), or consume/use data (Data Consumer, Data User, App Provider). In addition, Data Providers and Data Consumers may apply business models (including pricing models) as deemed appropriate.

- **Data Owner**: Is a legal entity or natural person creating data and/or executing control over it. This enables the Data Owner to define Data Usage Policies and provide access to its data.
- **Data Provider**: Makes data available for being exchanged between a Data Owner and a Data Consumer. As already mentioned above, the Data Provider is in most cases identical with the Data Owner, but not necessarily.
- **Data Consumer**: Receives data from a Data Provider. From a business process modeling perspective, the Data Consumer is the mirror entity of the Data Provider; the activities performed by the Data Consumer are therefore similar to the activities performed by the Data Provider.
- **Data User**: Similar to the Data Owner being the legal entity that has the legal control over its data, the Data User is the legal entity that has the legal right to use the data of a Data Owner as specified by the usage policy. In most cases, the Data User is identical with the Data Consumer.
- **App Provider**: App Providers develop Data Apps to be used in the International Data Spaces. To be deployable, a Data App has to be compliant with the system architecture of the International Data Spaces.

3.1.3.5 Intermediary

Intermediaries act as trusted entities. Roles assigned to this category are Broker Service Provider, Clearing House, Identity Provider, App Store, and Vocabulary Provider. These roles may be assumed only by trusted organizations. Benefit for participants in the International Data Spaces is created by these roles by establishing trust, providing metadata, and creating a business model around their services.

- **Broker Service Provider**: Is an intermediary that stores and manages information about the data sources available in the International Data Spaces.
- **Clearing House**: Is an intermediary that provides clearing and settlement services for all financial and data exchange transactions.
- **Identity Provider**: Should offer a service to create, maintain, manage, monitor, and validate identity information of and for participants in the International Data Spaces. This is imperative for secure operation of the International Data Spaces and to avoid unauthorized access to data.
- **Broker Service Provider**: Provides Data Apps. These are applications that can be deployed inside the Connector, the core technical component required for a participant to join the International Data Spaces. Data Apps facilitate data processing workflows.
- **Broker Service Provider**: Manages and offers vocabularies (i.e., ontologies, reference data models, or metadata elements) that can be used to annotate and describe datasets.

3.1.3.6 Software / Service Provider

This category comprises IT companies providing software and/or services (e.g., based on a software-as-a-service model) to the participants of the International Data Spaces.

- **Service Provider**: If a participant does not deploy the technical infrastructure required for participation in the International Data Spaces itself, it may transfer the data to be made available in the International Data Spaces to a Service Provider hosting the required infrastructure for other organizations.
- **Software Provider**: Provides software for implementing the functionality required by the International Data Spaces.

3.1.3.7 Governance Body

The Certification Body, Evaluation Facilities, and the International Data Spaces Association are the Governance Bodies of the International Data Spaces.

- **Certification Body and Evaluation Facilities**: The Certification Body, together with selected Evaluation Facilities, is in charge of the certification of the participants and the core technical components in the International Data Spaces. These Governance Bodies make sure that only compliant organizations are granted access to the trusted business ecosystem.
- **International Data Spaces Association**: The International Data Spaces Association is an organization promoting the continuous development of the International Data Spaces. More about the International Data Spaces Association in the next section.

3.1.3.8 The International Data Spaces Association

The International Data Spaces Association[1] (IDSA) was founded in 2016 as a non-profit organization promoting the continuous development of the International Data Spaces. More specifically, it supports the development of the Reference Architecture Model and the participant certification process. The IDSA aims to guarantee data sovereignty by an open, vendor-independent architecture for a peer-to-peer network which provides usage control of data from all domains.

3.1.4 TRUSTS Architecture

The architecture of TRUSTS consists of 3 core elements, the so-called nodes. There is the Central Node, the Corporate Node, and the User Portal Node. The Central Node serves as the central intermediary between the other two node types. With the help of the broker, contained in the Central Node, the users of the other nodes can retrieve metadata on available assets (datasets, services, and applications), which the companies provide via their Corporate Node. A user can search for and retrieve available assets in his User Portal Node with the help of a CKAN[2] interface. If a user wants to retrieve an asset, a contract is concluded between the user and the providing company, which contains conditions for the use of the asset. Compliance with these conditions can be checked with the help of the Dataspace Connector, which is included in every Node. Figure 3 gives an overview of the architecture.

[1] https://internationaldataspaces.org/
[2] https://ckan.org/

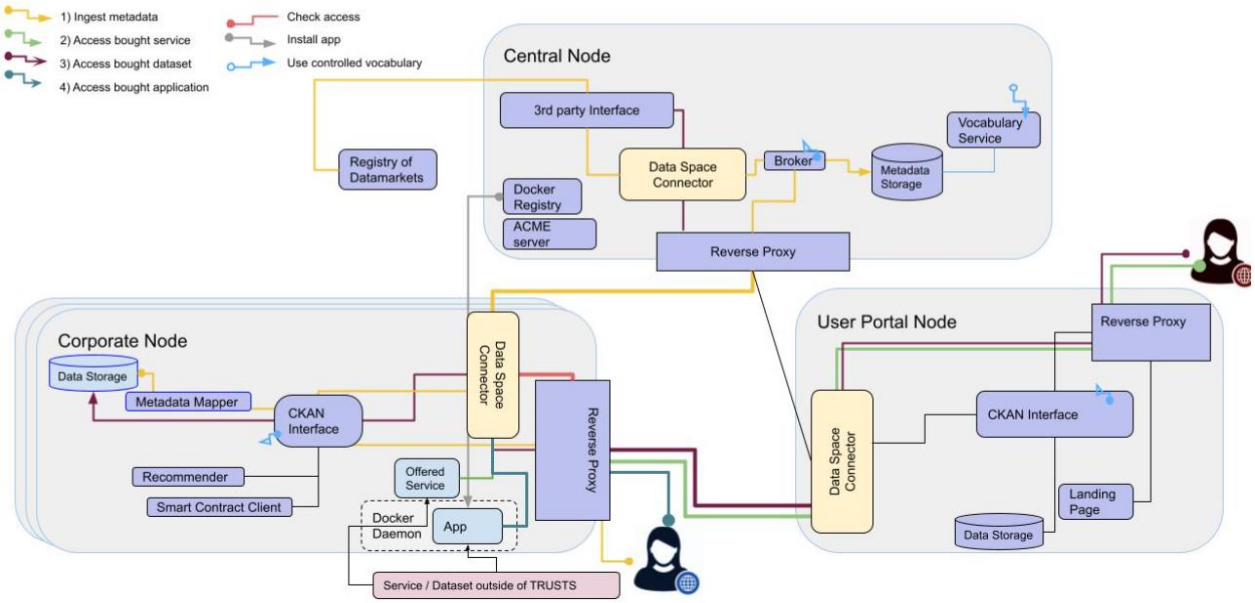

Figure 3: TRUSTS Architecture.

3.2 Blockchain

The blockchain is a decentralized, distributed, cooperative data store. It enables data to be exchanged securely in networks without the need for an intermediary. The basis of blockchain technology is the so-called "distributed ledger", a distributed logbook containing entries with information. These entries are time-stamped and also contain a reference to the previous entry. The entries, also called blocks, are tamper-proof by means of a cryptographic key - the hash. The fact that each new block contains a reference to the previous one creates a chain between the blocks - the "blockchain". An illustration of a blockchain is given in Figure 4. The first block in a chain is called the genesis block. In the event of a break in the hash sequence, this would immediately be identified as tampering. Due to the redundant storage of all data on all servers participating in the network, incorrect entries can be replaced [9].

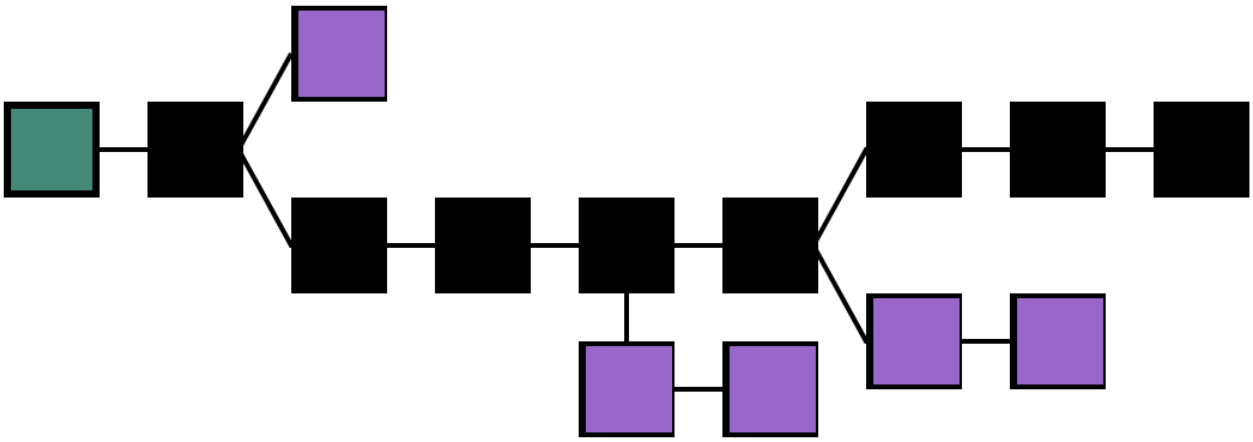

Figure 4: Illustration of a blockchain. The main chain (black) consists of the longest series of blocks from the genesis block (green) to the current block. Orphan blocks (purple) exist outside the main chain[3].

Irreversibility is a core characteristic of blockchain technology. This means that information or transactions in the blockchain can neither be manipulated nor deleted. The background to this is that the databases are no longer managed and stored centrally by an entity, but in a so-called peer-to-peer network (P2P network). The information in the database is stored on the computers of all participants, with entries being identical. If a participant creates an entry, it is checked, e.g., on compromise, by all other network participants and is not stored until they have all confirmed it. This mechanism is referred to as an integrated consensus mechanism, which, along with irreversibility, is one of the crucial trust-building properties of blockchain technology. In order to subsequently rewrite data once it has been stored, all of the participants' computers would have to be manipulated at the same time. The more participants there are in a network, the more difficult manipulation becomes. To reverse a transaction, the corresponding counter-reaction can only be stored in the blockchain by consensus. This also makes proofs of origin and transactions for mapped values audit-proof [10, 11].

Due to different objectives, there is a difference between public (Public) and non-public (Private) blockchains. Public blockchains are public networks that are primarily used for trading and exchanging cryptocurrencies. The Bitcoin blockchain is the oldest example. It is based on the white paper "Bitcoin: A Peer-to-Peer Electronic Cash System" published in 2008 under the pseudonym "Satoshi Nakamoto" [12].

In contrast, private blockchains are systems that are only accessible to a closed consortium. This can be, for example, several cooperating companies or organizations within a network [13]. In contrast to public blockchains, access rights to private blockchains are restricted to the consortium. If another participant is to be integrated into the network, this must be approved by the blockchain consortium. In addition, read, write and administration rights can usually be assigned individually for private blockchains. Administrability is of increasing importance in the context of industrial blockchain applications [10]. Examples of private blockchains are the "Hyperledger Fabric'', "Multichain" or "Corda".

The development of blockchain applications is often divided into three phases. If the focus is on the exchange of cryptocurrencies, blockchain solutions are grouped under the heading "Blockchain 1.0". The Bitcoin blockchain belongs to this group. The next major step that contributed to the development of

[3] Theymos from Bitcoin wikiVector: Razorbliss, CC BY 3.0 https://creativecommons.org/licenses/by/3.0, via Wikimedia Commons

blockchain technology was the introduction of smart contracts. All blockchain solutions that follow the smart contracting approach are classified under the "Blockchain 2.0" phase. An example of this is the Ethereum blockchain, which also belongs to the Public Blockchains. Unlike the first two phases, the term "Blockchain 3.0" covers all those applications that are not directly related to finance. Here, the focus is rather on decentralized, autonomous organizational units interacting on the basis of common smart contracts. An example of this is the Hyperledger Fabric mentioned above [10, 14] , which is the blockchain implementation chosen by the TRUSTS project for integration into the platform.

An example for such blockchains can be given in the supply chain management, where blockchains are being tested for automating payment transactions in value creation systems. This is done with the help of the smart contracts, described in the next chapter, which are based on the blockchain used. By eliminating manual activities, the processes involved are accelerated considerably, while the tamper-proof storage in the blockchain also ensures increased transparency throughout the entire blockchain network. Particularly in the international logistics environment, reconciliations, such as the Bill of Lading with the Letter of Credit, can be made more efficient and traceable [16], [17]. Other blockchain applications aim to store proofs of origin for goods transparently and securely and make them available to authorized third parties in a traceable manner. Thanks to the blockchain, goods can be traced seamlessly in trade here, effectively preventing fraud [20]. Furthermore, the blockchain also enables an increase in efficiency in the international movement of goods, which at the same time favors resource-saving processes. For example, in container trade and shipping in international maritime transport, confusing paper-based processes that previously had to be carried out manually can be mapped digitally. Here, the blockchain as an interoperable database between the involved partners offers the possibility to reduce the inefficiency of a paper-based way of working in this environment and to create a web interface for all verified participants [15, 18, 19]. Common to all the solutions presented are the goals of increasing transparency and reducing or eliminating inefficient processes. Blockchain technology raises some legal issues that need to be explored. For example, questions arise about how to stop the dissemination of illegal content in the blockchain or how to ensure IT security in the blockchain. If the blockchain network consists of participants in different countries, the question arises as to which legal system is applicable to the blockchain. Although transactions can be carried out via the blockchain in a legally secure manner, no legal claims can be established in the blockchain. This should be one focus of the investigations within the project.

3.3 Smart Contracts

The term smart contract was first introduced by Nick Szabo in 1994 and defined as follows:

“A Smart Contract is a computerized transaction protocol that executes the terms of a contract. The general objectives of smart contract design are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries.” [21]

Smart contracts are not contracts in the legal sense, but rather programmed if-then conditions. They can be seen as program codes that represent contracts in the sense of a simple if-then function and support their automated execution. If a certain event occurs, a predefined action is triggered. Szabo cites the example of a vending machine as something that embodies the characteristics of a smart contract. Once the money is paid, an irrevocable series of actions is set in motion. The money is retained, and a beverage is provided. In the process, neither the transaction can be stopped, nor the money returned when the

beverage is delivered. The terms of the transaction are in some sense embedded in the hardware and software that runs the machine [22].

Blockchains are particularly well suited for smart contracts because they not only store data in a tamper-proof manner, but also allow processes to be executed automatically. For example, temperature data can be fed into the blockchain at regular intervals via the Internet using a sensor attached to a refrigerated container. If damage occurs due to temperature differences, the chain of evidence is transparent via the blockchain. If the damage falls under the responsibility of a logistics service provider, the logistics service provider can pay out compensation immediately.

With the help of smart contracts, the partners in a blockchain network can define all the framework conditions of their relationships in a binding manner. As digital sets of rules, they are stored in the blockchain and are also automatically monitored by the system. When several cooperating companies have negotiated the terms of their collaboration and agreed on a set of rules, smart contracts can trigger the execution of agreed processes. For example, with the help of a smart contract, the payment process between two trading partners can be triggered automatically as soon as the recipient has received the goods. To do this, the information and goods flows between the companies must be recorded in a blockchain. The smart contract can then access this irreversibly stored information. If the goods were delivered in the agreed quantity and quality, this can be stored in the blockchain. The smart contract logs, which are also stored, can - after verification - trigger a predefined payment to the supplier based on this information, with the payment transaction itself also being stored in the blockchain. Thus, the administrative effort from invoicing to payment instruction can be significantly reduced, as these processes no longer must be carried out individually and manually but can be automated on the basis of the agreements made. Thus, smart contracts bring automation and intelligence to this system, in addition to data protection, while IoT sensors collect various data from the real world [23].

A major advantage is that no intermediaries are required between the contracting parties when using smart contracts. Whereas banks, law firms or notaries can ensure trust in the contract and check that all parties comply with the terms of the contract in the case of traditional contracts, there is no need for an additional, central intermediary in the case of smart contracts. Here, the various contracting parties rely on the processes stored in the program code and the automated exchange of services based on blockchain technology. The administrative effort and processing time for control on the part of an intermediary are therefore also reduced by its omission. As a result, the cost of monitoring contract compliance also decreases, as this is no longer handled by a central entity, but is instead carried out by the automated processes of the smart contracts. Once established, the smart contracts can automatically check the proper fulfillment of contractual obligations for each transaction and trigger or refuse the corresponding transactions [9].

Blockchains and smart contracts bring a variety of advantages, but they also have a set of drawbacks. These drawbacks will be more investigated in section 5.2.

4 Legal Challenges

4.1 Introduction to Smart Contracts

Smart contracts can be defined both from a feature-based (technical) perspective and from a functional perspective. Technically, smart contracts are a network of computer messages composed of conditional statements (i.e., "if condition "x" materializes itself, then effect "y" occurs") executed on a blockchain or distributed ledger technology (DLT). Contractual clauses are written in the form of code instead of human language, and hence their consequences are triggered as soon as the condition embedded in the clause materializes itself or not. From a functional perspective – and as the expression suggests – "smart contracts" are designed so as to enable automation of (part of) the execution of an agreement where automation refers to the computer-based execution without direct human intervention. [45]

Automation of part of a contract's lifecycle is not a new phenomenon. [46] The disruptive character of smart contracts rather lies in the fact that the latter are executed on a DLT: this means that the automation process is neither up to the parties nor up to a third entrusted party. As Ruskin states, "neither of the parties must trust the other for the contract to be performed. They must trust the disinterested blockchain, which can enforce the relevant terms. [45] via pre-defined algorithms that make legal consequences flow from the application of the contractual clauses written as code. In practice, this approach brings about a drastic change as both sides of the agreement's performance can be automatically processed by the smart contract. [46] Blockchain-based smart contracts can be defined as "a piece of software code, implemented on a blockchain platform, which ensures self-performance and the autonomous nature of its term, triggered by conditions defined in advance and applied to blockchain-titled assets". [46] Against this background smart contracts seem more efficient – with a view to the actual performance of the agreement to the benefit of the parties – than a traditional written contract where the parties would merely *commit* to perform. [47] By ensuring automated execution, smart contracts avoid non-performance of the agreement and thus seem to deprive recourse to judicial dispute-resolution processes of any relevance.

The first wave of the blockchain technology (Bitcoin and other cryptocurrencies) made it possible to execute transactions without relying on traditional financial institutions. Now, smart contracts (second wave of the blockchain technology) would make it possible to avoid relying on the judicial system to a large extent by guaranteeing enforcement of agreements by technical means. Not only does a smart contract make it possible to do without judicial authorities but it also enables the parties to make it impossible for any third party – including judicial authorities – to interfere in its execution notably owing to the immutable character of the blockchain: as legal scholars have pointed out, smart contracts illustrate Lessig's prediction that "code is law". [48]

Now there is no specific regulation of blockchain-based smart contracts in Europe or even at international level. However, it is crucial to stress that the EU institutions have explicitly recognized the importance of distributed ledger technologies and smart contracts. In 2020 the European Parliament acknowledged the increasing relevance of smart contracts and issued recommendations to the European Commission with a view to including in the upcoming Digital Services Act (DSA) specific provisions and criteria to determine the legality of smart contracts. [49] The Report highlights that the legal framework has not been keeping up with the technological and business developments leading to a rise in the use of smart contracts. In requesting the Commission to make proposals on a legal framework for smart contracts, the European

Parliament (i) stressed the need to propose provisions that clarify issues of legality, enforcement and use of smart contracts in cross-border situations; [49] (ii) stressed the need to ensure balance between parties to a smart contract, the need to ensure that public interest concerns, such as the targeting of anti-competitive cartel agreements; and (iii) and strongly advised to propose provisions that would allow halting and reversing the execution of smart contracts in certain circumstances. [49] The concerns expressed by the European Parliament are relevant to the goal of making smart contracts actionable in the current data economy and will be referred to throughout the chapter. [49]

As the current law stands, anyhow, there is no ad hoc framework regulating smart contracts, beyond some provisions of national law. For now, therefore, and with a view to issuing recommendations for such a framework, the relationship between smart contracts and the law needs to be tackled by the law regulating contracts, which includes standard contractual provisions and general principles of contract law. [50, 51] The section below highlights the key features of traditional legal contracts generally recognized by classic contract law and discusses whether such features apply also to smart contracts. This is done with a view to identifying the features of smart contracts that escape classic regulation and that pose a challenge from a legal perspective.

4.2 Applying Contract Law to Smart Contracts: Legal Challenges

A controversial issue is whether smart contracts belong to the typology of agreements regulated by classic contract law. [52] The issue stems from the fact that in the term 'smart contract' the word 'contract' is not intended to *necessarily* designate a contract in a legal sense: [53] In order for a given construct to qualify as a contract in a legal sense, it needs to present certain features that trigger the application of contract law rules. Because contract law is not harmonized and is largely based on the national laws of each EU Member State, there cannot be a single EU-wide answer to this question, and there is indeed no consensus within the literature to start with. [50, 54]

It appears that, because smart contracts are nothing more than lines of code that trigger consequences as soon as a condition occurs, the most promising approach to gauging the legal nature of smart contracts should not be trying to determine whether smart contracts can *systematically* be qualified as legal contracts. Rather, a more sensible avenue may be to be *a priori* neutral as to their legal qualification and apply a legal test on a case-by-case basis relying on the essential features of legal contracts. [54, 55]

This section recalls the entire lifecycle of contracts and explores the challenges that stem from applying contract law to smart contracts. From start to end, such challenges relate to: (a) the very existence of a smart contract in a legal sense and its validity; (b) if we deal with smart contracts qualifying legally as contracts, their enforceability and how it differs from classic contracts; (c) the interpretative issues generated by smart contracts compared to classic contracts: and (d) specific issues of cybersecurity and illegality typical of smart contracts.

4.2.1 Legal qualification and validity of smart contracts

Determining whether a smart contract can be qualified as a legal contract is an essential step to exploring subsequent questions related to the validity, enforceability, and interpretation of smart contracts. In this subsection we therefore first tackle: (a) the criteria for determining the legal qualification of a smart

contract; then, for those smart contracts that have a legal character, we tackle (b) the application of typical substantive rules on contract validity to smart contracts.

4.2.1.1 Legal qualification of smart contracts

As soon as conditional lines of code are written and embedded in a blockchain environment, it is indisputable that a smart contract exists from a technical standpoint. But whether it also exists in the realm of contract law is another matter. [56] In order to explore this question, it is useful to distinguish between smart contracts in a narrow sense - i.e., referring to nothing more than the lines of code written and meant to be executed in an automated way - and smart contracts in a broad sense - i.e., that takes into account the whole process of negotiating and concluding a contract before its automated execution.

We rely on the most influential legal systems of the EU and on the EU Draft Common Frame of Reference (DCFR), which tend to account for principles inspired by both the civil law and the common law traditions of EU Member States. [50] Looking at the DCFR, we see that for a contract to come into existence there needs to be the will of at least two parties to enter into an agreement and vest it with legal force for them. [57] This requirement presupposes two distinct though interrelated manifestations of active will from the parties: first, the will to enter into an agreement as such, i.e. the intention to make promises to one another; second, the will to make sure that the agreement does not merely qualify as an informal agreement grounded on social norms, but that becomes legally binding on all parties with the possibility for them to rely on applicable legal rules to demand its enforcement, and seek remedies for lack thereof. We explore each manifestation in detail.

The *intention to enter into an agreement* implies that neither party is forced to enter a contractual relationship (which also begs the question of authentic consent, see further below), and that neither party is brought onboard a contract unbeknownst to her. For a contract to exist all the parties to it must have triggered their authentic will to bargain; in other words, the formation of a nexus, a link between the parties needs to stem from each party's unaffected intents. This requirement is present, under varying forms, in the law of most Western jurisdictions, such as in German law, [58] French law, [59] Italian law [60] and English law. [61]

And in smart contracts?

In order to determine whether the above requirements can be satisfied by smart contracts, one needs to distinguish the whole experience of creating and executing a smart contract (broad sense) from the mere code embedded in the blockchain (narrow sense). Technically speaking, as the contract is executed automatically with no human interventions, the parties to it do not engage in any manifestation of will with a view to performing the contractual norms. Because of this, some authors argue that in smart contracts the machines replace the very act of 'promise' and 'will', [54] and that one of the distinguishing features of smart contracts is that the parties do not exchange any promise to one another. [45, 54]

However, their wills do manifest themselves when the parties agree to translate their intentions into code via a smart contract. Some smart contracts required the parties to agree upon it in advance, whether directly or through electronic agents used to that effect. [46] It can therefore be concluded that because the conclusion of a smart contract may not be automated, contrary to its performance and execution, the will of the parties can still be required before lines of code are created to substantiate the contract. Many smart contracts, however, simply materialize with a party proposing

the coded terms and another party accepting by triggering its performance, i.e., without pre-contractual negotiation culminating in an exchange of wills. [54] Because of this feature they are much closer to the concept of unilateral contracts. In this sense, much of the substance behind any manifestation of wills is likely to depend on the trend in smart contracts: the more standardized they will be, the less there will be a need for human negotiation, and hence the human element would be increasingly reduced. [54]

The *intention to attach legal consequences* to the agreement entails that the parties agree that their respective promises can be enforced on them by legal means. [62] This makes a contract an agreement with legal character. The parties agree to the existence of an *obligatio* (obligation) between them which means reciprocal rights and duties among them which – if not performed – may give rise to judicial actions by the counterparty (or even by public authorities in some cases). A contract is thus by essence forward-looking: the parties agree when concluding the contract to respectively execute a duty in the future or else – in a second step future – this may give rise to infringement procedure (see below); they agree to use the tool of contract to regulate the uncertainty of part of their relations. [63] This is referred to as animus *contrahendi* (latin for 'intention to enter into a contractual relationship'). The *animus contrahendi* condition is pursuant to the principle of freedom of contracting. [64] The typical example of an agreement deprived of legal character by lack of *animus contrahendi* is that of social agreement such as a dinner invitation. Another example – in the economic environment – is the conclusion of a "Gentlemen's agreement" [62] whereby trade partners make mere commitments guaranteed solely by their honor or reputation (excluding legal character). The right of the parties not to vest their agreement with legal character is, however, not infinite: whereas this right differs among national laws, it is at the very least clear that the parties cannot use this freedom to circumvent mandatory rules [63] whereas the latter depend on applicable law.

The above reasoning is not dissimilar to the doctrine of consideration, applicable in common law, which links the formation of a legally valid contract to an exchange of promises between the parties to be carried out via that contract. In this exchange the promisor shall reap a real benefit. [50] There is therefore an economic argument grounded in the formation of contracts in common law. From the element of 'exchange of promises' in the doctrine of consideration it is however clear that this element of reciprocity is more generally common to European contract law as a whole: in the contract law of most EU Member States, pursuant to the freedom of contracting and unless prohibited by the applicable law, the parties use legal contracts to regulate their respective substantial commitments one to another. In that sense, a contract is a legal act in that it creates legal obligations. [62] These commitments are legally binding so that they can be opposed in court by the aggrieved party. However, the content of the contract is legally binding only upon the parties to it pursuant to the principles of freedom of contracting, on the one hand, and of relativity of contracts, on the other hand. [63]

And in smart contracts?

Similarly to the box above, we need to distinguish the broad from the narrow view of smart contracts. In a narrow sense, smart contracts do not contain any legal obligations. They merely include lines of code dictating what will happen on the digital asset(s) being the subject matter of the contract once and if certain conditions materialize themselves. [54] The absence of obligations is a direct consequence of the way smart contracts are executed: because they are executed in an automated way, and not by humans, there is no need - and no use either - for a smart contract to contain an obligation for someone to do something. This description also applies to all smart contracts that are

merely proposed by an offeror and then accepted (or declined) by the other party with no prior interaction.

However, once again, the coded rules included in smart contracts can be the result of two or more parties agreeing on the performance of certain actions and agreeing to attach legal significance and legal effects to them. In this case, instead of relying on a mutual trust between them, the parties rely on the algorithm enabling the automated contract execution. [46] In this respect it is also worth pointing out that the element of 'consideration' required in common law contracts would be, in the smart contract's world, the performance itself: by their very nature, smart contracts imply the exchange of digital assets between two or more parties, which means that the material element of the doctrine of 'consideration' is always satisfied. [50]

Lastly, many jurisdictions based on the Romano-Germanic and the Anglo-Saxon legal families identify the conclusion of a contract when *an offer made by one party is accepted by the other party*. [50] This principle is also recognized in the DCFR. [63] As evidenced in the DCFR, the offer and acceptance scenario can be alternative to a scenario whereby two parties just exchange their wills to contract and come up with a set of legally enforceable rules without there being an offering and an accepting party. The opposite is not true, however: offer-and-acceptance types of contracts always imply the intention of both parties to enter the contract. The reality is simply more blurred than either archetype: some contracts are just offered and accepted with no negotiations (such as most public and private service contracts to consumers); others still entail an offer but are followed by negotiation as to the precise terms of the agreement.

Contract law foresees many ways through which a party's offer can be considered as accepted by another party. The basic scenario is to have one party explicitly communicating to the offering party that she accepts the offer. Different types of contracts call for different approaches to handling acceptance, such as: getting onboard a train equates accepting a public transport contract; inserting a coin in a vending machine equates accepting a contract for the sale of a good; etc. [63]

And in smart contracts?

The offer-and-acceptance mechanism can work in a smart contract scenario just as well as a typical meeting of wills of the parties, as explained in the previous point. The smart contract in a narrow sense - i.e., the coded conditions - is just the material medium through which the will of the parties is certified, and hence any traditional offer-and-acceptance dynamic that happens in the pre-contractual phase is compatible with a smart contract.

What is trickier is how to translate into a smart contract scenario the actual conclusion of the contract stemming from one party accepting the other party's offer. In other words, what action by the accepting party provides the necessary conditions to consider the initial offer accepted, and hence the contract concluded? How do offer and acceptance materialize themselves in the mechanics of a smart contract? Discussion on this point comes down to establishing *when* exactly a smart contract expected to be vested of legal character is concluded. This brings up the question of timing of consent, which is another way to look now in which the offer is met by an acceptance.

It could be argued that the concepts of 'offer' and 'acceptance' are attached to two of the key steps involved

in setting up a smart contract: deployment and call. The deployment - or 'posting' - of the smart contract on the blockchain by one user (the offeror) could be seen as the 'offer', since it is the step through which the offeror enables the other party to trigger the performance of the contract by calling its functions. In practice, the offeror publishes in the distributed registry of the blockchain a proposal to conclude a contract on a web page, generally signed with its private key, and accompanied by a link to the code. [50] Once the other party sends an electronic message, signed with its own key, she would be accepting the offer and the contract would be considered as concluded. [65]

This is however only one of the possible avenues that national contract law may take. Another one could be to consider the deployment of the smart contract as an invitation to engage into a contract, and not as a formal offer that triggers the other party's acceptance. [65] However, an argument can be made that the former approach makes more sense given the nature of smart contracts and their fundamental difference with traditional contracts in human language. The increased level of trust formally placed in smart contracts derives amongst other things from the immutable character of its content. In other words, once the logical connection between conditions and outcomes are written in code and deployed on the blockchain, there is no room for further negotiation. Because of this immutable character, it would make sense to equate 'deployment' with an offer, and not simply with a request to negotiate, because the deployment does not admit revisiting the terms of the contract. In this respect, therefore, by calling the smart contract in the blockchain the counterparty would be understood not merely as accepting the invitation to negotiate, but as accepting the terms of the contract, because through that call the execution of the smart contract would begin.

Some authors suggest that this perspective could be pushed as far as envisaging that the offer-and-acceptance mechanism could be completed even when the acceptance stage is not executed manually by one party, but electronically by the software based on a previous 'meeting of minds' between the parties. [50] Conversely, other authors hold that it is the 'mutuality' of the exchange that is missing in smart contracts and that deprives them of legal character. [66]

In any case, a contractual clause strictly preventing any right of recourse of (one of) the party(ies) would typically be null and void and deprived of any legal effect.

4.2.1.2 Validity of smart contracts

After establishing that smart contracts can belong to the realm of contracts in a legal sense from the perspective of the parties' will to engage in a contractual relationship, we shall examine whether smart contracts can satisfy substantive and formal rules related to contract validity. We first examine substantive rules of contract validity, and in particular rules related to: (a) valid consent and vitiated consent; and (b) automated or manual activation of contracts.

Valid and vitiated consent are issues that are closely related to the existence of the will of the parties. If a party is willing to agree on specific terms, she is ready to give her consent to those terms. Consent is deemed to be valid when it is freely given, without any manipulation or coercion by the other party or a third party. In such a situation the initial will of the party concerned corresponds to her consent to the actual terms of the agreement. Conversely, consent is vitiated whenever a factor intervenes that makes

one party give consent whereas she would most likely not absent that factor. Examples are mistakes as to facts or law; [63] incorrect information or 'misrepresentation'; [67] fraud [68, 69]; and violence or threat. [70, 71]

Depending on the applicable law, different factors are considered as vitiating one party's consent to a contract, and different legal consequences are drawn from establishing that a certain factor played a role.

And in smart contracts?

We saw earlier that a pre-contractual phase can certainly precede the conclusion of a contract that is translated into a smart contract. In theory, therefore, applicable rules related to valid and vitiated consent would apply to such a pre-contractual phase. However, it is likely to be much more challenging to (a) ascertain instances of vitiated consent to smart contracts; and (b) draw the necessary consequences.

First, smart contracts are written in computer language, and it may be argued whether each party to the contract fully understands the terms she engages to. That is likely to depend on the extent to which the contract is negotiated in prose before being written in the form of code. Factors such as mistakes as to law or facts; inaccurate communication; and incorrect information are therefore likely to play a role, but it may not be straightforward to ascertain them.

Second, even if one were to ascertain that any of the above factors vitiated one party's consent, the extent to which consequences can be drawn from this leads to an unsettled debate, i.e., the degree of permeability of the smart contracts realm to the traditional system of remedies (see more on this further below).

Information duties may mitigate this challenge. For example, to the extent that a given smart contract falls within the meaning of 'distance' or 'off-premises contract' referred to in Article 6 of the Consumer Rights Directive, [72] a consumer would be entitled to require adequate disclosure of information in the pre-contractual phase. This is likely to be very relevant in case of contracts involving individuals and parties such as banking and financial institutions; companies providing targeted marketing services; etc. From a general point of view, some authors call for a balance between, one the one hand, duties to disclose as much information as possible in the pre-contractual phase, and on the other hand, a presumption as to the unvitiated consent of the parties to a smart contract. [74]

The last element concerning substantive requirements of validity is about the 'entry into force' of smart contracts between the parties and the fact that smart contracts can also be *activated in an automated manner* through the intervention of oracles, i.e., differently than the case where one party activates them by performing a call. [74] This appears to be an issue specific to smart contracts and with no equivalent in traditional contract law since it stems directly from the technical infrastructure that makes automated activation of smart contracts possible. It is not difficult to conceive why one may be discouraged from recognizing automated activation as a legal method for exchanging consents: the automatism is machine-driven, which is something *prima facie* alien to the very concept of 'consent', which should be expressed by a human act. However, this argument risks offering a short-sighted view of the smart contract flow: the automated activation does not substitute itself to the free will of the parties involved, which, as we saw

above, can be expressed in an adjusted version of the offer-and-acceptance mechanism at work in traditional contracts whereby the acceptance is manifested differently than with a call by one party.

In this sense, the automated activation is just a technical enabler that facilitates execution. Therefore, to not recognize automated activation through oracles as complying with the requirement of free and unvitiated consent might unduly limit technical progress and uptake of smart contracts in the data economy. Legal uncertainty may arise insofar as different jurisdictions take different approaches to this question.

4.2.2 Enforceability

Any discussion on the enforceability of smart contracts needs to acknowledge that the approach to enforcing smart contracts is inherently different to the approach to enforcing traditional contracts. This has to do with the infrastructure sitting behind smart contracts: differently from traditional contracts, which generally require action from one or more parties to execute, smart contracts execute automatically upon the materialization of the conditions and circumstances linked to the envisaged outcomes, without the need for the parties to have recourse to traditional enforcement mechanisms. These mechanics work thanks to the blockchain. A smart contract includes logically interconnected statements that are expected to be triggered when a given condition is produced. As soon as this happens, the corresponding statements in the smart contract are activated, which translates into transactions being executed and validated in the blockchain. [75]

The blockchain mechanics are in principle suited to solving legal and practical challenges linked to contract enforcement: from the legal point of view, automated contract execution can reduce drastically the instances of unpunished breach of contract, potentially leading to higher legal certainty between commercial partners and spreading higher trust that may encourage businesses to enter commercial transactions. [76] Both *bona fide* and voluntary breaches are prevented: because the performance of smart contracts does not depend on the parties' own performance, the execution is not subject to human error or subjective discretion; at the same time, because the performance of smart contracts does not depend on the willingness of the parties to abide by their terms, smart contracts prevent opportunistic behavior, such as 'efficient breach'. [46]

From a more practical standpoint, because of higher compliance with contractual terms, the parties are likely to spend less time and resources in contract performance monitoring, lawsuits, and remedial actions. [46, 56, 75, 77] Authors also suggest that smart contracts have the potential to improve and make more efficient even traditional enforcement mechanisms, such as online dispute resolution (ODR). [77]

4.2.2.1 Consequences and challenges related to immutability

The blockchain-based nature of smart contracts has a consequence on the content of such contracts: once the terms are validated in the blockchain, they are in principle immutable, unlike in traditional legal contracts whose terms may be amended subject to both parties' agreement. [54] As paradoxical as it may sound, the immutable character of smart contracts may be both beneficial and detrimental to legal certainty. As we just saw, immutability *benefits* the legal certainty of the parties about the execution of the contract. It increases trust as the parties know that a) in principle the terms agreed to are not going to change and b) they are going to be executed automatically by a technology (the blockchain) which does not depend on either party. [45] This works for the legal order created by a given contract between the parties. However, this legal order is not the only one to consider. To the extent that the parties want the

smart contract to have legal effects for them, the contract does not operate in a vacuum, but within a much larger legal order that is all but immutable.

Depending on the subject-matter of a smart contract, provisions of administrative law and private law may apply to the very terms agreed to by the parties. This triggers two issues: first, the need to ensure *pre-deployment compliance* before creating the smart contract; and second, the need to ensure *continuous compliance* of the smart contract. We tackle both issues in succession.

As to *pre-deployment compliance*, in principle it is an easier problem to tackle than continuous compliance because it requires the usual due diligence of the parties involved in a pre-contractual relationship vis-à-vis the law, making sure their terms are in keeping with the applicable legal framework. This requires that the parties observe their duties and, where necessary, are appropriately counseled. The main challenges concerning pre-deployment compliance relate to the transformation of terms expressed in human language into code, which we discuss in the next section.

As to *continuous compliance*, it is easy to see why the immutable character of smart contracts may be *detrimental* to legal certainty: the smarter contracts are shielded from the evolving legal framework that applies to them, the more the parties risk being kept in the dark as to whether or not the terms they agreed to are (still) permissible. Due to the inherent functioning of contractual terms validation in a blockchain and to the technical difficulties to update smart contract code, [45] to update a smart contract after its deployment is not an easy task, yet it is arguable that it is necessary to foresee a way to do so, especially to the extent that smart contracts become increasingly relied upon in all sort of transactions, including data transfer agreements and provision of data processing services such as those envisaged in data marketplaces such as TRUSTS. It is arguable that it is desirable that this kind of transactions are not merely construed as 'gentlemen's agreements or agreements deprived of legal force, because, with the expectedly increasing volume of such transactions, such an approach could eventually lead to conducting most data-related transactions in a legal vacuum. It is desirable that this type of transactions is regulated by smart contracts concluded with the intention to legally bind the parties to them and, for this reason, the smart contract phenomenon calls for a solution to ensuring continuous compliance. The most investigated solutions so far are two: a) allowing updates by the parties *ex post*; and b) allowing updates through publicly available legal databases. [45] We discuss each approach below.

An approach that *allowed updates by the parties ex post* would require little to no large-scale infrastructural investments, as each set of parties would be called to 'police' their own contracts. This approach, however, has several downsides. First, from a legal policy perspective, it relies heavily on the parties' willingness, preparedness, and responsiveness to incorporate novel legal provisions into their contracts. Whilst the pre-contractual phase already implies to a certain extent trusting that the parties will not agree to illegal terms in the first place, it is likely that not all parties will be as diligent about the legal framework in force du laid out ring contract performance as they may have been during contract formation. Moreover, whilst the deployment of a contract can be subject to a certain degree of scrutiny capable of spotting potential illegalities, no such filter would exist during contract performance. This approach is therefore likely to lead to fragmented continuous compliance. Second, the simple possibility for the parties to amend the terms of the contract ex post is precisely what the concept of blockchain-based smart contract intends to avoid, i.e., interference of the parties with contract execution and increase in trust on the terms agreed to. [45] To be sure, such interference would be warranted by the need to comply with new law, but how to ensure that the parties do in fact limit themselves to this? It would require mutual monitoring and potentially lead to uncertainty, which is exactly what smart contracts are expected to reduce.

Differently, an approach that *allowed updates through publicly available legal databases* would likely materialize itself in a public infrastructure to which smart contracts would be connected, most likely via Application Programming Interfaces (APIs). This solution in the EU context would require substantial effort to bring together the official journals of the EU and of all Member States into a constantly updated electronic database. National and EU contract law would be enough for purely internal smart contracts, but cross-border transactions, such as those likely to be concluded in the context of TRUSTS and similar data marketplaces, would require access to legal updates in multiple jurisdictions. Then the system should be constructed in such a way that the smart contract automatically updates its code after 'calling' the relevant updates shared in the form of code in the database. Such a solution has the downside that it requires all smart contracts with legal force to be connected to a database encompassing the whole EU and that needs to be timely and reliable to share legal updates. However, such infrastructure could in principle be envisaged in the framework of the recent and future EU initiatives promoting digitalization in the realm of cross-border civil and commercial justice. [78]

4.2.2.2 Remedial action

The two aforementioned approaches could contribute to minimizing the extent of the divergence between smart contracts and the applicable legal framework. Moreover, contract drafting techniques, such as those that envisage sufficiently vague provisions to accommodate possible legal developments, may also contribute to reducing legal tensions to a minimum. However, the possibility that smart contracts lead to illegal outcomes or that require ex post enforcement exists and needs to be considered. We sketch here how the law could respond to this need through a system of remedies, which would not be focused on enabling one or more parties to seek redress (in the form of pecuniary compensation, specific performance, etc.) for a breach of contract by another party [45] - as in traditional contract law - but rather on remedying outcomes that are against the law. [45]

First off, it is worth reminding that any remedial strategy would need to be an *ex post* strategy, since by definition the remedy would be imposed upon the materialization of a given outcome. Part of the literature argues that ex post dispute resolution is the only phase where the human connection can occur in smart contracts. [54] While the pre-contractual phase, as we showed above, can also be humancentric, it is true that the human intervention to fix outcomes produced by machines can be especially problematic. This calls into question the competence of judicial authorities. The legal character of the contract implies a right of recourse of the parties before judicial authorities (and sometimes administrative authorities) to have the contract enforced and to resolve disputes related to it. This entails that the counterparty is not entitled to circumvent this right of recourse. As a rule, the parties have a certain leeway to regulate this matter by contractual means to a degree depending on the subject matters and on the branch of law concerned. For instance, they may as a rule disregard courts' authority to the benefit of legally recognized alternative dispute resolution means (e.g., arbitration courts). This uncertainty is reflected in some authors' argument that it is still too early to envisage what course of action public authorities would choose since the smart contract phenomenon is barely starting, [45] and still much in its infancy especially in the EU digital internal market.

4.2.3 Language and interpretation

As stated repeatedly, one of the most striking differences between smart contracts and traditional contracts is that smart contracts are written in the form of code. Smart contracts can also be represented in natural language - and, after all, they are the coded result of negotiations that happen in natural language - but such representation would be merely auxiliary to the language embedded in the smart

contract validated in the blockchain. The self-sufficiency of the code can be interpreted as meaning that the code itself is the ultimate arbiter of the smart contract, and that smart contracts are not meant to be interpreted by agents coming from the outside. [46] This feature would benefit interpretation because the less ambiguous the contractual terms, the smaller the room for disputes and the need for ex-post interpretative tools. [45, 51, 73]

However, this does not mean that smart contracts can render the topic of contract interpretation entirely irrelevant. Part of the literature even argues that smart contracts are by nature incomplete and ambiguous precisely because they are based on code. [79] There are indeed two caveats worth exploring: a) first, because the smart contract may well reflect a given will that the parties have formed in the real world, negotiating on the basis of natural human language, the question arises as to how to translate into code non-binary legal terminology, such as 'reasonableness', or 'good faith'; and in general, the issue is to make sure that the code can adequately capture the nuances of the parties' will; b) second, the code of the smart contract may include errors, potentially also due to misunderstandings between the 'abstract' formation of the contractual terms and their translation into code. Code may also be subject to bugs and flaws. [46] We tackle both points in succession.

4.2.3.1 The challenge of translating human legal language into code

Smart contracts are the first example of contracts whose execution is based on terms written in a language other than human language. Computer code is inherently different from human language not really because of its material manifestation, but because it is deterministic and conceived to convey just one given meaning, whereas human language is rarely shielded from ambiguity and the need for interpretation. [51, 79] Legal language being a subset of human language, it has been observed that, in order to shield contracts from the courts' interpretative leeway, common law contracts are generally more detailed and hence less ambiguous than civil law contracts, usually shorter and much more reliant on the application of general principles of contract law in court. [79] This has led some authors to hold that smart contract language is in principle closer to the language of common law contracts than civil law contracts. [79]

Computer code, however, remains inherently different. In accordance with the conditions-based structure of smart contracts, it is particularly suited to drafting in the form of "if condition 'a' applies, then consequence 'b' occurs". This might highlight a potential tension: while common law contracts and smart contracts tend towards the same outcome - i.e., be self-explanatory and reduce the scope for judicial interpretation - they do so via two very different approaches: because they are based on human language, common law contracts need to be very long and detailed to foresee any possible element; conversely, smart contracts tend to be very short successions of conditions. Hence the questions: are smart contracts - hence: code - in actuality suited to regulating very complex contractual relationships between the parties? Are smart contracts only suited to regulating basic transactions, or can they be rather easily built and used to replace the complexity of potentially any traditional contract? These questions trigger the ability of smart contract code to capture the nuances commonly found in contracts based on human language.

One situation in which this ability may not be straightforwardly available is the translation of typical civil law concepts into smart contract code. The concepts of 'reasonable efforts'; 'proportionate means'; 'good faith'; 'best endeavors', etc. are examples of notions that, to be operationalized, need the recourse to human judgment and appreciation, which sit at odds with computer code. In civil law jurisdictions the conclusion of a contract also involves for the parties the duty to comply with the law applicable to their agreement even without any mention in the contract: [63, 80] the most fundamental complementary legal duty is the duty of good faith [63] which applies throughout the lifecycle of the contract. [51] The content

of the duty of good faith as well as the sanctions in case of breach depend on the national law, on the branch of law at stake and on the quality of the contracting parties. [63] In civil law jurisdictions the duty of good faith is considered as a complementary duty of the party when performing its contractual duty. It also usually entails the obligation for the aggrieved party to mitigate its damage or at least not to aggravate it. A contrario breach of the duty of good faith may in certain circumstances qualify for an abuse of right on behalf of the creditor.

How to include these principles in smart contracts-based transactions? Legal scholarship has not reached a consensus on this issue, and it is not even clear whether such a principle should have any bearing on the smart contracts' ecosystem. [51, 52, 81] Attempts to 'force' these principles into smart contracts may turn out to be inefficient and costly, to the detriment of the utility of smart contracts. [52] In this sense, smart contracts may lead to the appearance of a code-driven bias, i.e., that instead of attempting to make the code fit the parties' will, the parties will need to fit the constraints of computer language. [51] If this happens to be the prevailing trend, there is also the risk that smart contracts either start to be used as an alternative to the legal order - which, as mentioned above, is not a desirable outcome because it would deprive many parties, especially vulnerable ones, of the protection granted by the law; or are confined to regulating only those transactions that do not present too high a level of complexity. In line with the European Parliament's recommendations, some authors highly recommend drafting ad hoc legal rules to capture smart contracts to avoid undesirable outcomes and to exploit the potential of the phenomenon in terms of legal certainty, enforceability and interpretation. [50, 51] The code-driven bias mentioned above, however, may only be one outcome of the tension between natural and computer language: the more drafters realize how difficult it is to translate commonly used legal terms into code, the parties may increasingly become aware of the ambiguity of natural language and be incentivized to better define their intentions. [79] In other words, the constraints of smart contracts may drive precision and clarity from the very source of the agreement.

Coming back to general principles of contract law, it can be argued that such principles, even when they are not formally coded into the contract, still do affect the lifecycle of a smart contract in two ways: a) first, they regulate, to an extent that depends on the principle at issue, the pre-contractual relationship between the parties, i.e. the phase before the contractual terms are drafted into code; b) second, when the law of the land recognizes them as part of general contract law, these principles ought - at least in theory - to be complied with during contract performance or the smart contract will be executing outside the realm of the law. However, this begs the question of how such compliance should occur given that smart contracts self-perform. Simply put, the execution *strictu sensu* of smart contracts cannot be affected by such general principles because it is an immediate, condition-based operation triggered by computers. 'Good faith' is an inherently human principle that loses all its meaning when the execution does not even depend on the will of a human being.

Because of the above challenges, one suggestion is to accompany the smart contract with a human language 'equivalent', which enables non-expert third parties - such as courts, should it be needed - to understand the contractual terms. [74] It can be argued that such human language representation may even be necessary, especially in the early stages, also to the parties themselves. It is not possible to conceive that all parties likely to engage in a smart contract have the technical skills to agree to coded terms before having them written down 'on paper'.

4.2.3.2 Challenges and implications stemming from coding errors or ambiguities

The above paragraphs have tackled a situation whereby there is an objective difficulty in translating terms coined to be used by humans into a language used by computers. This difficulty may result in a discrepancy between the will of the parties and the contract. However, discrepancies may also result from less visible

inconsistencies, such as when the party making available the smart contract assumes to be translating the will of the parties correctly and in its entirety, but in fact does not; or when the coding process exposes errors or flaws. Because there exists no ad hoc legal framework regulating smart contracts, such an occurrence is not accounted for in current law and leaves much uncertainty. [51, 54]

Moreover, we should also consider that the very premise of the above discussion concerning the switch from natural to computer language - i.e., that code is unambiguous - is not entirely true. At first glance, because computer code is based on conventions that always produce the expected result, it may seem that they are shielded from ambiguity. But as Grimmelmann highlights, the conventions on which computer code is based are themselves a social construct - but the 'social element' appears below the surface, whereas it is starkly evident in natural language. [79] Regardless, we would argue - as the author himself does - that computer language is a step towards effectively reducing ambiguity, if only because the meaning of linguistic constructs in computer language is far more 'finite' than any equivalent in natural language. [79] As long as the social constructs behind the programming language chosen to implement smart contracts hold, smart contracts will continue to be less ambiguous than contracts drafted in human language.

Carrying over one proposal from the former paragraph, one solution could be to envisage programs that translate natural language into code and vice-versa. [51] Although prone to errors themselves, such programs could nonetheless reduce the margin of error because the translation would be made by computers and not by humans.

4.2.4 Cybersecurity risks

Cybersecurity risks are amongst the potential downsides of smart contracts, or at least the areas that may decrease their effectiveness and uptake. [52] Cyber risks are generally different depending on the blockchain model chosen, i.e., public (or permissionless) blockchain vs. private (or permissioned) blockchain.

Public blockchain are based on shared computational power and, with no safeguards, the party or parties that come to control most of the computation power could theoretically affect the validation process guaranteed by the blockchain itself and rewrite its content. [56] Advanced security methods, such as proof of stake consensus, are possible remedies to the vulnerability of public blockchains. [56] Private blockchains try to solve this issue by adopting a zero-knowledge proof model, whose logic is not too dissimilar to the concept of multi-party computation in that the system can ensure the validity of a smart contract without disclosing information about the parties and other transaction details. [82] This capability relies on public parameters. Should someone be able to retrieve the random numbers those parameters were obtained from, they may compromise the security of the whole system. [82]

From the legal standpoint cyber risks are also particularly relevant. First, cyber risks threaten compliance with data protection principles, in particular data security. In this sense, blockchains do not present risks that other typical cyber infrastructure has, such as a single point of failure, because the copies of the ledgers are conceived so that, if one is hacked, this will not affect the other copies. [74] This feature reduces the risk of exposing the data to unauthorized parties, thus contributing to implementing the data security principle. [83] However, the risks sketched above with regard to public or private blockchain models need to be addressed to systematically uphold this principle.

Second, cyber risks may be problematic from a legal policy point of view to the extent that they threaten the good faith of the parties to the contract and undermine legal certainty. If a given cyber risk were to be

found particularly significant and common, it could threaten the uptake of the technology but in parallel also make it more difficult for the technology to uphold the principle of legal certainty in the realm of contract law through smart contracts. The increase in legal certainty promised by an ideally near-perfect functioning of smart contracts would remain illusionary. It is too early to evaluate whether cyber risks concerning blockchain-based smart contracts have the potential to become that significant, but if they were to be, the consequences on the legal system smart contracts are embedded in would need to be closely monitored.

5 Smart Contracts in TRUSTS

5.1 Data Transactions and Smart Contracts in TRUSTS

Transactions in a data market are carried out between two participants. Generally, data is exchanged within a transaction. However, it is also possible to offer various services via a data market, such as the one implemented with the help of TRUSTS components. As described in the architecture (Figure 3), there are nodes for users on the one hand and nodes for companies on the other, which are used by the various participant groups to communicate with each other. Furthermore, the architecture contains an additional node, which is described as the Central Node. This node contains various components, such as a broker, through which users can find data records and services from companies. Once a user has found a suitable data set or service via the CKAN interface of his node, they can conclude a contract with the corresponding company and use this data set or service in accordance with the contract.

In addition to the broker, the Central Node has other components that require central hosting by the data marketplace. These components should include, among others, a blockchain. It is used to enable the execution of smart contracts within the data marketplace. All participants who can interact with the blockchain are also in a so-called blockchain network.

As previously explained in chapter 3.4, the participants in a data market can use smart contracts in such a network to define all the framework conditions of their relationships in a binding manner. As digital sets of rules, they are stored in the blockchain and are also automatically monitored by the system. When several participants have negotiated the terms of their cooperation and agreed on a set of rules, smart contracts can trigger the execution of agreed processes. The process is illustrated in Figure 5. For example, a smart contract can be used to automatically trigger the payment process between two participants once the recipient has received the record or used the service. To do this, the information flows between the participants must be recorded in a blockchain. The smart contract can then access this irrevocably stored information. If the data record was transmitted to the participant, this can be stored in the blockchain. The smart contract's logs, which are also stored, can - after verification - trigger a predefined payment to the supplier based on this information, with the payment transaction itself also being stored in the blockchain. In this way, the administrative effort from invoicing to payment instruction can be significantly reduced, as these processes no longer have to be carried out individually and manually but can be automated on the basis of the agreements made. Smart contracts thus bring automation and intelligence to this system in addition to data protection [29].

Figure 5: Smart Contract Lifecycle.

Another advantage is that no intermediaries are required between the contracting parties when using smart contracts. Here, the various contracting parties rely on the processes stored in the program code and the automated exchange of services based on blockchain technology. The administrative effort and processing time for control on the part of an intermediary are thus also reduced by its omission. As a result, the effort required to monitor contract compliance is also reduced, as this is no longer handled by a central office but by the automated processes of the smart contracts. Once set up, the smart contracts can automatically check the proper fulfillment of contractual obligations for each transaction and trigger or reject the corresponding transactions [9].

Within TRUSTS, this is an essential aspect, as it allows the fulfillment of policies for a contract to be guaranteed. This also benefits the use cases of TRUSTS, in which data exchange and the use of third-party services play an important role.

To be able to describe the technical implementation of a blockchain component with smart contract execution more in detail, the requirements must first be explained more closely. This is done in the subsequent section.

5.1.1 Requirements

For the technical implementation of a blockchain component with smart contract execution, requirements for the components are defined below, through which they can be used in a TRUSTS-based data market. The requirements are divided into two categories: functional requirements and technical requirements. The latter can be derived from the functional requirements.

5.1.1.1 Functional Requirements

The functional requirements for the technical implementation of a blockchain component with smart contract execution are described in deliverables D2.2[4] and D2.3[5]. The first requirement is FR10: The system should provide contract mechanisms as a validation means of sellers'/buyers' agreements. The FR states that the component should provide contract validation capabilities to allow participants in a transaction to have their agreement validated. The next functional requirement, FR11, states that "The system should ensure the integrity and authenticity of the smart contracts transactions signed by its users. In other words, the system must ensure that when a participant executes a transaction, the integrity and authenticity of that transaction can be guaranteed using smart contracts. FR12 states that: Smart contracts in the system should be accompanied by a human friendly representation (i.e., natural language). This means that the smart contracts must be able to be reflected in a human-readable contract in natural language. FR14 specifies: The system should encompass mechanisms for keeping transactions from being infringed. So, transactions must not be violated or broken by any of the participants. The system should provide mechanisms for this. The last functional requirement, FR15, states: The system should provide the ability to connect to billing mechanisms for enabling consumers to pay providers according to the agreed smart contract.

[4] https://www.trusts-data.eu/wp-content/uploads/2020/10/D2.2-Industry-specific-requirements-analysis-definition-of-the-vertical-E2E-data-marketplace-functionality-and-use-cases-definition-I.pdf

[5] https://www.trusts-data.eu/wp-content/uploads/2022/01/D2.3_Industry-specific-requirements-analysis-definition-of-the-vertical-E2E-data-marketplace-functionality-and-use-cases-definition-II_Dec2021.pdf

These functional requirements specify various conditions that should be met in order to operate a blockchain component with smart contract execution in TRUSTS. In the following section, technical requirements are derived from the functional requirements.

5.1.1.2 Technical Requirements

Various technical requirements can be derived from the functional requirements and technical characteristics of blockchain technologies, which are listed and described in more detail below in Table 1. The requirements reflect the foundation for a technical implementation.

Table 1: Technical Requirements.

Requirement	Explanation
Blockchain	The blockchain is the central component when it comes to executing smart contracts. Through various technologies, which a blockchain has by definition, it is possible to fulfill FR10, FR11 and FR14. In addition, the blockchain must be able to execute smart contracts.
Permission	There are two variants of how the blockchain can be operated: public (permissionless) and private (permissioned). Public blockchains are, for example, the Bitcoin or the Ethereum blockchain, to which everyone has access. Private blockchains are used in delimited networks, such as companies, where only a limited number of participants are allowed to interact with the blockchain [84].
Smart Contracts	Smart contracts must be defined in advance by the parties of a blockchain network. For this purpose, contracts must be agreed upon at the beginning. Subsequently, these can then be transferred into the so-called chaincode [87]. This allows FR12 to be fulfilled.
Type	There are two types for smart contracts: On-Chain & Off-Chain [27, 28]. Each type offers its advantages and disadvantages, which must be considered for the technical implementation of a blockchain component with smart contract execution.
Node	The blockchain has consensus mechanisms that ensure the integrity of the blockchain while adding a new block to the chain. This satisfies for example FR14. However, the realization of a consensus mechanism is only possible if multiple instances of the blockchain are operating on different nodes within the data market or blockchain network [88].

Figure 6 graphically represents the process of adding a new block to the blockchain. In the initial situation, subscriber A wants to send subscriber B a data set. All the information from A's transaction is then written to a new block. Before the new block is appended to the blockchain, a copy of the block is sent to all blockchain instances in the network. Consensus mechanisms are used to validate the new block. This ensures that the entire chain is still valid afterwards. The new block can therefore be hung up after the chain, which means that the transaction has been carried out and participant A can send participant B the data record. The consensus mechanism is used in steps 3 and 4 in the figure.

The next section explains a concept for the technical implementation of a blockchain component with smart contract execution.

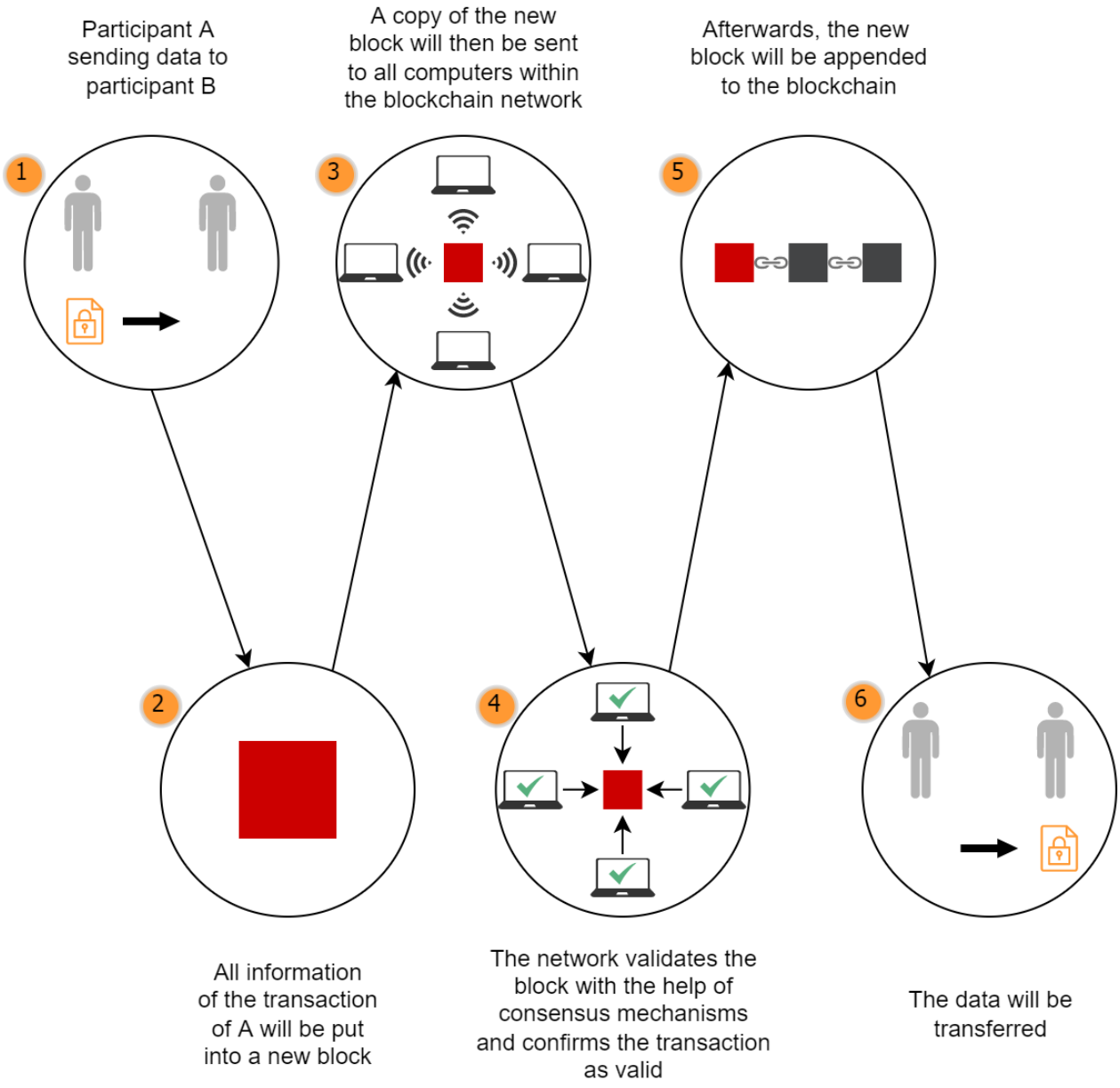

Figure 6: Adding a new block to the blockchain while an asset shall be transferred.

5.1.2 Technical Implementation

For the technical implementation of a blockchain component with smart contract execution, the choice of blockchain technology is primarily crucial. There are several technologies that can be used: Bitcoin, Ethereum, MultiChain, Tendermint, Hyperledger Fabric, Lisk, etc. Based on the functional and technical requirements, an initial selection can already be made here. The decisive factor for satisfying the permission requirement is that a permissioned blockchain technology, i.e., a private blockchain, is used. A data marketplace based on TRUSTS components is generally a closed marketplace. Participants need an access authorization in the form of a certificate to participate in the marketplace. Each participant is thus

identifiable. In addition, a private blockchain has the advantage that only one or more administrative instances are authorized and not all instances of the blockchain have the same rights [9]. This increases the security in the blockchain network from manipulation of the consensus mechanism. To further increase security, as shown in Figure 6, the blockchain component should be run on multiple nodes within the blockchain network. This can ensure the successful execution of the consensus mechanism, making it more difficult to tamper with the blockchain.

As already described in the technical requirements, smart contracts templates must be defined in advance and adapted to the corresponding human-readable contracts before they can be implemented in the respective chaincode of the blockchain. There are two types of smart contract implementations, which are described in more detail in the following Table 2.

Table 2: Chaincode types.

Type	Description
On-Chain	The smart contracts are executed on a runtime environment of the blockchain. They can be viewed by all, as they themselves are also stored on the blockchain.
Off-Chain	They are stored outside the blockchain. The smart contracts are also executed outside the blockchain. However, their process is not necessarily traceable. [27, 28]

Since TRUSTS is a closed data market, an on-chain approach should be chosen, especially in view of the high security requirements. Although this entails performance losses in the event of a large number of simultaneous transactions, the blockchain network retains full control over the smart contracts that are available on the network, and which are executed on the blockchain.

For the technical implementation of a blockchain component with smart contract execution that meets the various requirements in TRUSTS, *Hyperledger Fabric*[6] is the most suitable blockchain technology. Hyperledger fabric is an open source blockchain technology under the Linux Foundation. It is based on a modular and permissioned architecture and is thus capable of operating as a private blockchain. It supports the execution of smart contracts, which can be developed for any application domain, regardless of the solution model. For example, the domains can be in the context of finance or data exchange. Hyperledger Fabric has low latency, which allows created transactions to be executed quickly, and has various consensus mechanisms that ensure the integrity of the blockchain. In addition, the private channel is an important feature included in blockchain technology to ensure security and privacy within a blockchain network. Only participants in a channel can see their own transactions, which in turn means that a TRUSTS data marketplace can only be operated with them.

5.1.3 Use in a TRUSTS data market

As explained earlier, several Hyperledger Fabric instances are to be operated as separate and independent components in the data market. In order to be able to store transactions on the blockchain and trigger smart contracts, TRUSTS requires a component that serves the blockchain interface. Referring to the TRUSTS architecture (see Figure 3), CKAN is, besides others, an appropriate tool for this purpose. CKAN is

[6] https://www.hyperledger.org/wp-content/uploads/2020/03/hyperledger_fabric_whitepaper.pdf

a data catalog which helps organizations publish and manage their records and services, as well as search for records and services (resources). Via the user interface, once a user has found a suitable resource in its own CKAN UI, the user can negotiate a contract with the provider. Once an agreement has been reached, the user can use its Dataspace Connector (DSC) to request the resource. The DSC connects to the corporate node where the physical location of the resource is. A DSC within the corporate node responses to this request. This second DSC uses, in this example, the corporate CKAN instance to get the resource. But before CKAN is returning any resource, it connects to the blockchain via its Smart Contract Client and uses the Hyperledger Fabric's API.

The Hyperledger Fabric's API is to create a transaction on the blockchain that includes the terms of the request. CKAN is then able to verify these terms by triggering an appropriate smart contract. So, depending on the usage and contract, the company who runs the corporate node triggers appropriate smart contracts on the blockchain using transactions to verify the terms of the contract. For example, if a user and a company have agreed that the user may use a service from the company 10 times for a fee X, the company can create a transaction each time the user requests the service, which in turn triggers the execution of a matching smart contract on the blockchain. This contract verifies whether the user is authorized to use the service or not. The feedback from the smart contract is then transmitted back to the company. As a result, the company can allow or deny the user to use the service. Figure 7 illustrates the process with a sequence diagram.

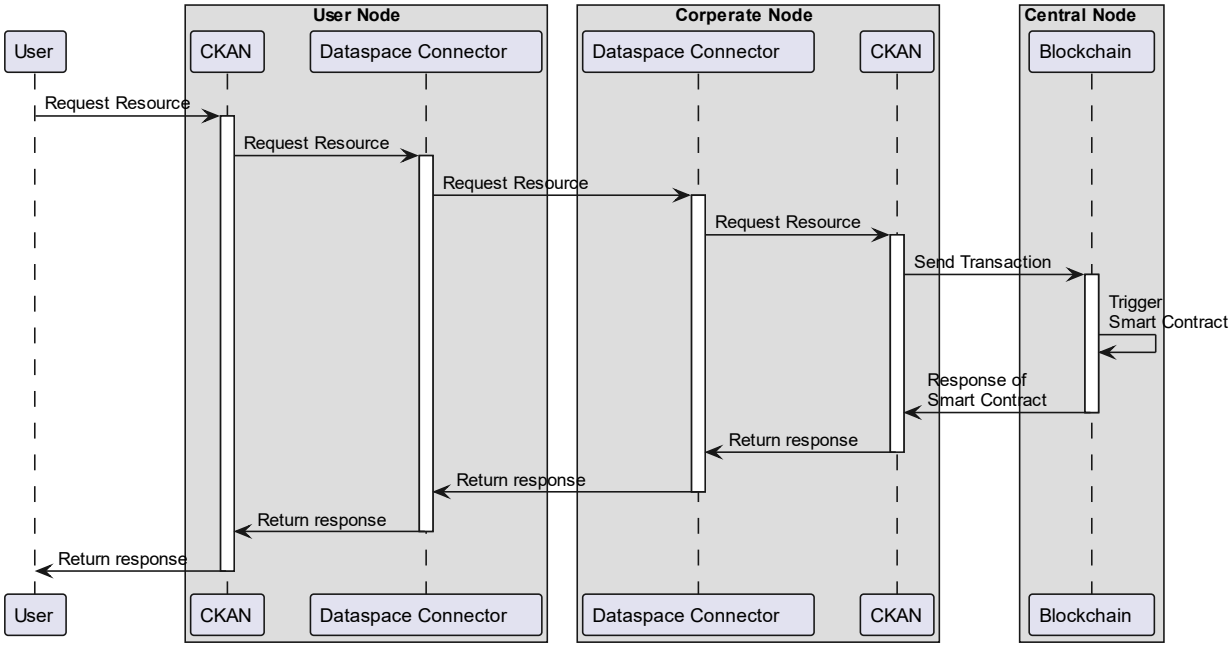

Figure 7: Trigger a smart contract

This section uses definitions and examples to define requirements and formulate a technical implementation of a blockchain component with smart contract execution, as well as describing the application in the TRUSTS data marketplace. In the following section, the risks and disadvantages of smart

contracts are examined in more detail. Chapter 0 provides an overview of privacy and security for smart contracts, in which various security mechanisms are examined before chapter 7 presents the practical implementation of this concept by means of a smart contract demonstrator.

5.2 Complications of Smart Contracts in TRUSTS

In this section, several issues, risks, defects, and drawbacks of blockchains and smart contracts are identified that can arise when a blockchain is operated and participants take part in a blockchain network.

Risks and Drawbacks

The following list is based on the work of Christidis and Devetsikiotis [9] and describes several risks and drawbacks using blockchains and smart contracts in practice. Each of the issues is described in more detail in the following Table 3.

Table 3: Risks and drawbacks using smart contracts.

Risk	Description
Lower Performing	Compared to a properly configured central database solution, a blockchain solution is typically lower performing. This results in higher response times and lower throughput when processing transactions. In addition, adding a new block cannot be parallelized because each node performs the same task. This situation is even more pronounced in blockchains that use smart contracts.
Privacy	Maintaining privacy in the blockchain is a complicated issue. Each participant in a blockchain network is identifiable by its public key (or its hash). A participant does not need to know the key from every other participant, only the key of its transaction partner. However, all transactions in a blockchain take place visibly for all partners involved in the blockchain network. By analyzing this data, an interested party can identify patterns and make connections between addresses, eventually drawing informed conclusions about the actual identities behind them. There are some options to mitigate, but not eliminate, this problem if privacy is important to the application in question.
Compromised	Another aspect to consider when setting up (or participating in) a blockchain network is the formation of groups that can compromise the blockchain. A participant is not able to forge a transaction or rewrite the history of the Blockchain. However, a participant can prevent a new, valid transaction from being added to the blockchain. This is generally prevented by the consensus mechanism, however, the number of nodes deciding on the validity of a transaction is usually significantly large. However, if several participants join together to form a group and the size of the group reaches a certain threshold up to which a consensus mechanism can prevent the compromise of the blockchain, the risk increases that transactions will be censored by being rejected by the group and thus not ending up in the blockchain. The nodes of the participants to be decided must therefore be chosen carefully to minimize the likelihood of collusion between them.

	In a private network, therefore, legally binding contracts should be signed in advance so that collusion among participants is appropriately penalized.
Legal Enforceability	The legal enforceability of smart contracts is limited. Work is ongoing to make the technical rules of smart contracts legally enforceable and binding on all parties. Until then, what happens if, despite the verifiability of the entire process, a transacting entity doubts the outcome of a smart contract operation? One way to increase the chances of legal enforceability is to include a reference to the actual real contract in the smart contract and vice versa. This process is called "double integration." In the event of a dispute, you can reference the hash stored in the smart contract, then present the real contract (the authenticity of which can be verified) and prove the link between the actions on the blockchain and the expected outcome in the physical world.
Full Autonomy	Full autonomy is a double-edged sword: before deploying a smart contract on the chain, one should carefully consider its logic; one should also include fail-safe mechanisms in the code to avoid deadlocks. There may be smart contracts whose overall behavior can change based on user input. Or there may be a feature that allows a privileged user to delete the provisioned contract and remove it from the blockchain. However, if none of these precautions are taken, we are dealing with a system that can never be changed. This in itself is not a bad thing. But if a function in that contract is misspelled, all interactions with it cannot be undone. A simple example is a smart contract that is supposed to act as a depot box on the chain. One can deposit funds (units of a cryptocurrency) and also withdraw them. The person who implemented it on the blockchain did not provide fail-safe measures, such as a 'self-destruct' function that allows the contract to be removed and its funds confiscated. If the 'withdraw' function of the contract is misspelled (by mistake), all funds deposited into the depot box are irrevocably gone and cannot be retrieved.

Defects

In addition, Chen et al [86] discussed 20 defects that can occur in smart contracts. These were divided into 5 categories. The presentation of the defects goes into detail, so only a few defects are presented in Table 4 and only given a rough explanation. The authors focused their research for the paper on the blockchain Ethereum. However, the defects can also be applied to other blockchains with smart contract execution, as the functionalities are similar.

Table 4: Defects using smart contracts.

Defect	Description
Unchecked External Calls	When calling functions in other smart contracts, network errors may occur, for example. Callback functions should take effect at this point, but their return values must also be taken into account by the caller to ensure the code logic.
Unmatched Type Assignment	As with other programming languages, it is important to choose the right data type. This makes the execution of a smart contract more efficient on the one hand and more secure on the other.

Transaction State Dependency	Smart contracts must validate whether the person who called the contract is authorized to call it or certain functions in the contract at all. If the verification is incorrect, this can have serious consequences.
Unused Statement	As in classical programming languages outside of smart contracts, the following applies: unneeded variables and statements should be removed.
Hard Coded Address	Since smart contracts cannot be modified after their deployment, hard-coded addresses in such contracts can lead to vulnerabilities.

The complications highlighted must be taken into account when operating a blockchain component with smart contract execution in TRUSTS in order to increase security for the data market and prevent data misuse. In the following chapter, security aspects and data protection of smart contracts are considered in more depth and various security principles are presented.

6 Smart Contract Privacy and Security

This chapter focuses on the privacy, security, and integrity of smart contracts. The privacy subchapter goes into detail regarding related privacy preserving technologies applicable to smart contract systems and provides some context on terms such as privacy and confidentiality. The security subchapter describes security principles, common vulnerabilities and past practical attacks related to smart contract and blockchain systems. The integrity subchapter provides some discussion on how integrity is enforced in smart contract and blockchain systems, as well as how the introduction of privacy preserving technologies can complicate matters for integrity.

6.1 Privacy of Smart Contracts

This subchapter covers the privacy of smart contracts, including a contextualization of the area and terms such as privacy and confidentiality, as well as private and consortium blockchains, along with a description of some of the more prevalent privacy preserving technologies in the area of smart contracts.

6.1.1 Privacy Preserving Smart Contracts

A smart contract, when executed as a blockchain application, must operate within the context of said blockchain. Blockchains were not designed to cater to data privacy concerns. The blockchain was intended to be used to verify integrity and legitimacy using a chain of transactions that is rooted in strong cryptographic hashes. All data on such a blockchain is totally available, in unencrypted form, to all of the blockchain's participants. This provides a substantial obstacle for applications which possess a data privacy requirement.

6.1.1.1 Private and Consortium Blockchain Variants

Alternative variations on the blockchain have been developed to address privacy concerns. The private blockchain model opts for a more centralized approach to that of the original blockchain model; the private blockchain is managed by a single organization or authority. Enforcement of the privacy of operations as well as the ability to alter past transactions are benefits of the private blockchain. The consortium blockchain is quite like the private blockchain, with the chief difference being that it is managed by multiple organizations. Both private and consortium blockchain cases need to enforce data privacy somehow; sensitive data cannot be visible to all members of a blockchain, even a private or consortium variant.

6.1.2 Smart Contract Privacy Vs Confidentiality

Privacy, as the term pertains to smart contracts, describes the protection of sensitive data against unwanted access by actors who should not be able view, consume, or buy it, as per the wishes or rights of the data owner. Many techniques are employed to enforce data privacy, such as zero-knowledge proofs and cryptography.

Confidentiality, as related to smart contracts, can be described as a scenario between a data provider and a data processor. The data provider allows a data processor, and only that data processor, to hold their data and to carry out some consented processing actions. For example, the data processor sharing the

data provider's artifacts with a third party would violate confidentiality, whereas carrying out processing or actions which were not agreed upon would not violate confidentiality per se but would instead constitute misuse of the data. Violation of a data provider's data confidentiality in a scenario such as this is, of course, intrinsically linked to a violation of the privacy of that same data.

6.1.3 Overview of Privacy Preserving Technologies for Smart Contracts

This section provides an overview of privacy preserving technologies in smart contracts and includes content from the Dell EMC whitepaper "Blockchain for Off-Chain Smart Contracts in an SAP environment - Trustworthy processing of private data …" [32] to include some of Dell's past solution ideas in this area. This section also provides content taken from the current literature in this area.

6.1.3.1 Encrypted On-Chain Data with Homomorphic Encryption

This approach involves the storing of sensitive data on-chain, in encrypted form. Given that the data is encrypted, a smart contract application cannot process it without being able to view the data in its unencrypted form. This presents the issue that, when the decryption operation is performed, all participants on the blockchain will have visibility of that operation which compromises the cryptographic keys used and the data itself thereafter. In order for this approach to work the smart contract would have to process the data while it is still in encrypted form by leveraging homomorphic encryption [33]. At the time of writing this, homomorphic encryption, though performant enough for small to medium scale applications, in a large-scale enterprise-level scenario efficiency may be a concern if using this technique, though it presents a strong solution in the future even at this scale.

6.1.3.2 Secure Multi-Party Computation

Secure Multi-Party Computation, or sMPC, is a cryptographic approach that can be applied to Privacy-Preserving Smart Contracts. The sMPC approach uses the standpoint that if sensitive data is split into specific subsets, so as to ensure that each subset is meaningless by itself, that computations can be done by separate actors on each individual subset of the data. The results of these computations can then be combined to provide the given output, with the actors doing the computing never having known the meaning of the input data [34]. In this case the input data can be seen as a Privacy-Preserving Smart Contract which requires some computation done.

Although theoretically extremely resistant to attacks (every single one of the input computing actors must collude to ascertain the meaning of the input), sMPC is not, presently, a fully practical approach - this is partly due to the fact that a communications bottleneck occurs if too many participants are present.

6.1.3.3 Secure Enclave

A Secure Enclave [41] conceals the state of a computer program and prevents external access to it, cutting-off the enclaved code from outside espionage or tampering. This secure enclave is provided by a Trusted Execution Environment (TEE), such as Intel SGX. The combination of Secure Enclaves with Asymmetric Key Cryptography provides one the ability to encrypt a Smart Contract using the Secure Enclave's Public Key. Given that the corresponding Private key exists within the Secure Enclave the ciphertext of the Smart Contract can then be sent to that Secure Enclave, where decryption and processing of the Smart Contract can occur.

The chief issue with Secure Enclaves to date is that the Intel SGX chip is by far the leading hardware in terms of Trusted Execution Environment technology. This creates a heavy reliance for TEE on a single chip architecture; an issue compounded by a handful of recent practical attacks on SGX, such as those known

as *Meltdown* and *Spectre*. The argument against the Trusted Execution Environment is that it is a hardware-based approach, rather than a cryptographic and mathematics-based approach, with the implicit assumption in this argument being that a trusted hardware guarantee is not as resistant to attack [36].

6.1.3.4 Zero Knowledge Proofs

A Zero Knowledge Proof works by carrying out a validation of an item without revealing underlying information about the item being checked. An example of this is a challenge presented by the verifier to the prover, where the prover can only reliably pass the challenge if they hold specific knowledge of some information, and where the verifier can perform this verification while knowing nothing about this underlying information [35].

One example of implementing Zero-Knowledge Proofs is by use of Cryptographic Hashes. A Hash value is created when input data is run through a Hashing Algorithm, creating a unique, uniform output that is virtually impossible to reverse-compute in order to ascertain the input data. The input data in this scenario can be any facet of Smart Contract Data, or Metadata. Once selected this input data can be Hashed and sent to the verifier, or if the verifier for some reason should not have access to the underlying data, it can be sent to a Trusted Third Party for verification. Such a scenario may occur, for example, in proving to a service that one is the resident of a particular country without wishing to send personal identifying information, such as passport details, to the service.

This Trusted Third Party (for example a government body) holds the Provers' confidential data, in this case the smart contract, and can therefore verify whether a hash has been computed using elements of that same confidential data. The Trusted Third Party can then notify the Verifier of whether the Hash Value provided by the Prover was legitimate or not, subsequently determining whether the Prover holds the input data in question. The more of these verifications that are computed, the smaller the probability that the prover has "gotten lucky", or somehow ascertained a single artifact from the confidential data.

6.2 Security of Smart Contracts

In this subchapter is detail regarding the main principles of security in smart contracts, as well as a list of common smart contract vulnerabilities, based on past practical attacks against smart contract and blockchain systems. A matrix diagram provides a cross-comparison of platforms and past attacks. This diagram is by no means all-encompassing but serves to provide an overview of the kinds of attacks that have occurred in the past against prominent blockchain and smart contract systems.

6.2.1 Smart Contract Security Principles

Smart Contracts are an emergent technology, and their development does not conform to the same principles as development of more common software applications. The high-impact nature associated with smart contract threats (financially or reputationally) suggests smart contract development is more akin to other development areas with high-impact threats, such as software for aviation, rather than areas like web application development. Four important principles [37] to bear in mind during smart-contract system development are as follows:

The first of these principles, testing, is ubiquitous to broader software development; testing the smart contract software thoroughly and iteratively to account for vectors as they arise. Phased testing of updates using increasing workloads and user-bases are widely used to make for a smoother final rollout of an update. In less critical systems it may be more difficult to justify more stringent testing, however in the context of smart contracts expending this additional effort in the test phase is preferable to a potentially impactful threat coming to fruition.

Robust error and exception handling is essential to smart contract software. The principle is not to adopt the mindset that no error should ever occur (which is impossible) but instead to accept that errors are an inevitability of software development, and to build resilient software which can deal with issues when they arise. A competent system for deploying patches and updates is essential to this principle; no matter how well the software handles an error or exception - that software issue will still need to be fixed.

Complexity of smart contracts; the smart contract software should be as simple as possible. Additional complexity in code is linked with the chance of program errors occurring. Part of avoiding overcomplexity lies in following a programming paradigm to produce compact, modular code which isolates functional blocks. Using the blockchain too liberally when it is simply not needed should be avoided - use of the blockchain should be reserved to work items where it is essential. Leveraging existing products and applications rather than writing functionality in-house is also favorable where possible.

Updating; any off-the shelf tools or applications must, of course, be kept at the latest release version. Rapid patching of security flaws or bugs as soon as they become known is prudent, as is researching the latest developments in cyber-security to ensure that the best techniques available are being employed to protect the smart contract software, but also because news of security flaws can often be made public due to the exploitation of a zero-day vulnerability.

6.2.2 Common Smart Contract Vulnerabilities

Smart Contracts, similar to any software, are not immune to design and programmatic flaws which can create an attack surface which can be exploited to perform a multitude of malicious actions. Source code errors and program bugs are a key contributor to smart contract vulnerabilities arising. If standard programming languages have been utilized to develop the smart contracts or some of their surrounding services, then any inherent security risks in those programming languages are inherited by the smart contract. Virtual machines also come with their own array of risks and compromising a virtual machine on a blockchain network can allow an attacker to target many assets on that network, smart contracts included. This section will address vulnerabilities which are, or have been, commonly used to attack smart contract systems. Table 5 synopsizes some of these smart contract attacks on leading blockchain networks [37, 38, 39, 40]. The important point here is that Hyperledger Fabric is the development kit that the TRUSTS blockchain was created with, meaning that the denial of service and remote code execution attacks could be of greater importance to this architecture. More information is provided on these common vulnerabilities in the following subsections.

Table 5: Smart Contract attacks on Blockchain Networks [37, 38, 39, 40]

	Ethereum	EOS	NEO	Hyperledger Fabric
Denial Of Service	✓		✓	✓
Overflow	✓	✓		
Storage Injection			✓	
Remote Code Execution		✓		✓
RAM Exploit		✓		
Re-Entrancy	✓			

6.2.2.1 Denial of Service Attacks

A denial-of-service attack [42], as the name would suggest, results in service being denied to legitimate users. Here the term "service" can describe access to any desired system component. Resource starvation by flooding a system component with requests can block users from accessing said resource, or in extreme cases can even crash the target system. This concept can be applied to smart contracts by calling a specific function repeatedly to tie it up for as much time as possible, reducing access to it for legitimate users. To increase the impact of the attack the functions which incur a significant amount of processing time on the system handling the requests are targeted.

Bandwidth starvation is another type of denial-of-service attack which operates by sending a flood of packets across a specific network which is used by or related to a target system. The intent is to consume so much of the available bandwidth that legitimate users are unable to contact the target system. This type of network bandwidth attack is typically more effective against centralized systems; distributed systems necessitate a distributed attack. If the attacker can target a vital system resource, for example a system storing smart contracts off-chain (as described in [32]), then the effects of degrading the service of this key component could, in turn, impact the entire system's ability to carry out smart contract operations.

The prospect of an attack based on a malicious smart contract has been explored in the past. Typically, this type of attack preys on a logical or program language vulnerability. A malicious smart contract is effectively a computer virus and can create enormous impact. In terms of what could be done using malicious smart contracts ranges from eavesdropping to data tampering and most of everything which can be carried out on a system by a traditional computer virus. One such real-world vulnerability which was discovered would result in a specifically coded malicious smart contract crashing any system which processes it. A further real-world vulnerability was the "log4j vulnerability[7]" which resulted in an attack vector for denial-of-service attacks [31].

[7] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44832

6.2.2.2 Overflows - Integer and Numeric

An integer overflow sees a stored integer variable occupy an excess amount of the memory address register - specifically more than it is assigned. In this instance the overflow is executed by assigning an integer data type with a value larger than it can normally store. This technique exploits a lack of boundary value analysis during operations involving arithmetic logic. Successfully manipulating arithmetic logic in this case would create a value larger than the underlying structure (database column, datatype) can hold.

This should produce an error or exception from the software. Where such overflows are not handled, or boundary values not properly analyzed is where the attack can find purchase. A simple, yet effective impact of such an attack is the consumption of memory; consuming enough to starve underlying systems of necessary resources and thereby reduce or eliminate availability of services to users - which, furthermore, could just be the initial step in a more complex attack.

6.2.2.3 Storage Injection

In a scenario where a "total amount" variable is accessible by multiple data types (string, integer, etc.), and where robust checks for validity do not exist, a clear entry point is presented to attackers. Where a function takes in three parameters: sender, receiver, amount, coupled with the aforementioned lack of robust checks, one could pass invalid input in a given parameter. In this scenario the attacker could only perform relatively benign actions such as spending their own currency to alter the "total amount" variable. The crucial point here is that this is a storage injection attack which can modify underlying data types without permission. To envision how this may be a more impactful attack one need only change one of the parameters to something like "account number", "username" or "password".

6.2.2.4 Remote Code Execution

This vulnerability results from an error in the underlying functionality libraries on a given platform; in this case with the code that handles the important task of array bounds checking. This vulnerability is exploited by use of malicious smart contract programs. This malicious contract's unbounded array values are written to spurious memory address registers of the target system which circumvents security mechanisms such as Address Space Layout Randomization and Data Execution Prevention. This vulnerability allows the attacker to write into any memory address, including restricted memory on the target system/ The consequence of this is that the attacker can, among other nefarious actions; carry out a privilege escalation, such as promoting their user account to an administrator, or alternatively (or even additionally) setup a remote connection via SSH to their own machine - commandeering the target system. A real-world example of this is a "log4j vulnerability" which affected any system using the log4j library, including the fabric gateway software [30] and provided an attack vector for remote code execution [43].

6.2.2.5 RAM Exploit

The RAM exploit described in this section was leveraged to create a resource starvation attack by allocating such large amounts of RAM on the target system that there was an insufficient amount available to perform legitimate user operations. The mechanism by which the RAM is consumed in this case is via a malicious smart contract which is specifically engineered to occupy RAM, for example by generating much garbage output or by causing a memory leak [44].

The fraudulent smart contract is sent to a victim who signs it based on its face-value transaction that they wish to consent to. A vulnerability in the system means that their signature is instead utilized by the attacker to authenticate a different operation entirely, for example the malicious smart contract using a "double-free" exploit to free the same RAM allocation twice in order to cause corruption in the RAM allocations. During the supposedly legitimate operation (tokens being sent) the attacker could take

advantage of the corrupted RAM allocations and insert rows into the victim's data table, which would be filled with garbage data and result in the allocation of their available RAM, which would amount to deliberately causing a memory leak.

This could be exacerbated by the fact that, on some target smart contract networks, RAM which has a process allocated to it cannot be deallocated until said process has finished. In such instances the garbage output memory leak caused by the malicious smart contract means that the processes would tie up the ram allocation indefinitely - until some form of effective intervention happens.

6.2.2.6 Re-Entrancy Attack

A re-entrancy attack sees the attacker perform a slew of calls to a specific function, where subsequent calls are made prior to the completion of the process started by the initial call in the slew. Successful re-entrancy attacks in the past have utilized this process on a system which has vulnerabilities in its checks - namely that a balance check would be performed at the initial step of each contract; one could initiate many initial steps without completing a transaction - thus priming a set of processes to execute their subsequent processes in parallel which could be used for nefarious purposes such as moving more units of a cryptocurrency than is permitted by the contract, or rather by any single instance of a check on a given contract. Clearly the situation presented here can be avoided by modifying the design logic of the system; updating the balance at the initial stage of the operation or putting some kind of hold or reservation on the balance in question would counteract the effectiveness of re-entrancy duplication of balance in the described scenario.

6.2.2.7 Updating On-chain Smart Contracts

The immutability of the blockchain makes it difficult to update on-chain smart contracts, for example to upgrade them or resolve security issues and program bugs. This can result in blockchains running outdated, and potentially vulnerable, versions of smart contracts unless alternative means are used to make necessary changes. A simple way of doing this is to install the newer smart contract version to the blockchain and block the older one programmatically, for example by refusing requests from older versions, meaning that although the older version smart contract remains on the chain that nobody can use it. Another notable alternative solution to this is to use an on-chain pointer or proxy which directs the user to a smart contract program that is off-chain. This allows for the off-chain, backend smart contract to be updated as necessary and the on-chain address to remain the same. Hyperledger Fabric, which was the blockchain platform we chose for the TRUSTS smart contract demonstrator, provides this kind of functionality. A Hyperledger chaincode, or smart contract, can run off-chain in a secured and isolated container.

6.3 Integrity of Smart Contracts

A means by which the integrity of smart contract software can be assured to all to all participants in the operation is to first give code visibility to those concerned, allowing them to generate a cryptographic hash of the contract if they so choose. The contract software which is run can then be subject to the same hashing mechanism; thus, providing the participant with an assurance rooted in cryptography. The blockchain is an ideal mechanism for the described integrity checks due to its verifiable chain of transactions.

The introduction of a privacy preserving smart contract complicates this matter somewhat, particularly if dealing with off-chain smart contracts [32].

A scenario where the participants in the smart contract operation do not consent to the sharing of their smart contract input data with the other participants, rather only the result of the operation is to be shared is one example of such possible complication. This would necessitate a privacy preserving solution - and furthermore negate the storing of human-readable format smart contracts on-chain.

Off-chain smart contracts can cooperate with a blockchain by having their hash value uploaded to the blockchain. Of course, some participants may want additional proof that the smart contract instance running during the current operation is indeed valid, and that a malicious user has not tampered with the software since said hash code was placed on the chain.

Certain trusted execution environments are capable of both preserving privacy - by running the smart contract in an enclave, as well as providing participants with cryptographically rooted assurances of the integrity of the smart contract program; that it is indeed identical to that which they have previously validated. Just as important is the ability of trusted execution environments to prove to external entities that they are indeed a bonafide and legitimate trusted execution environment. This is known as an “attestation protocol”.

7 Smart Contract Demonstrator

The term “smart contract demonstrator” is the name assigned by Task 3.2 to a tightly scoped, technical addition to the task’s existing written output, namely a system which enables the TRUSTS project to utilize blockchain functionality, as well as smart contract functionality over an API, which was developed as part of this work. The architecture of the smart contract demonstrator is shown in Figure 8, which provides an architectural overview of the component itself and shows the linkage between various subcomponents and other TRUSTS platform components and clients – all of which will be described in detail in the following sections.

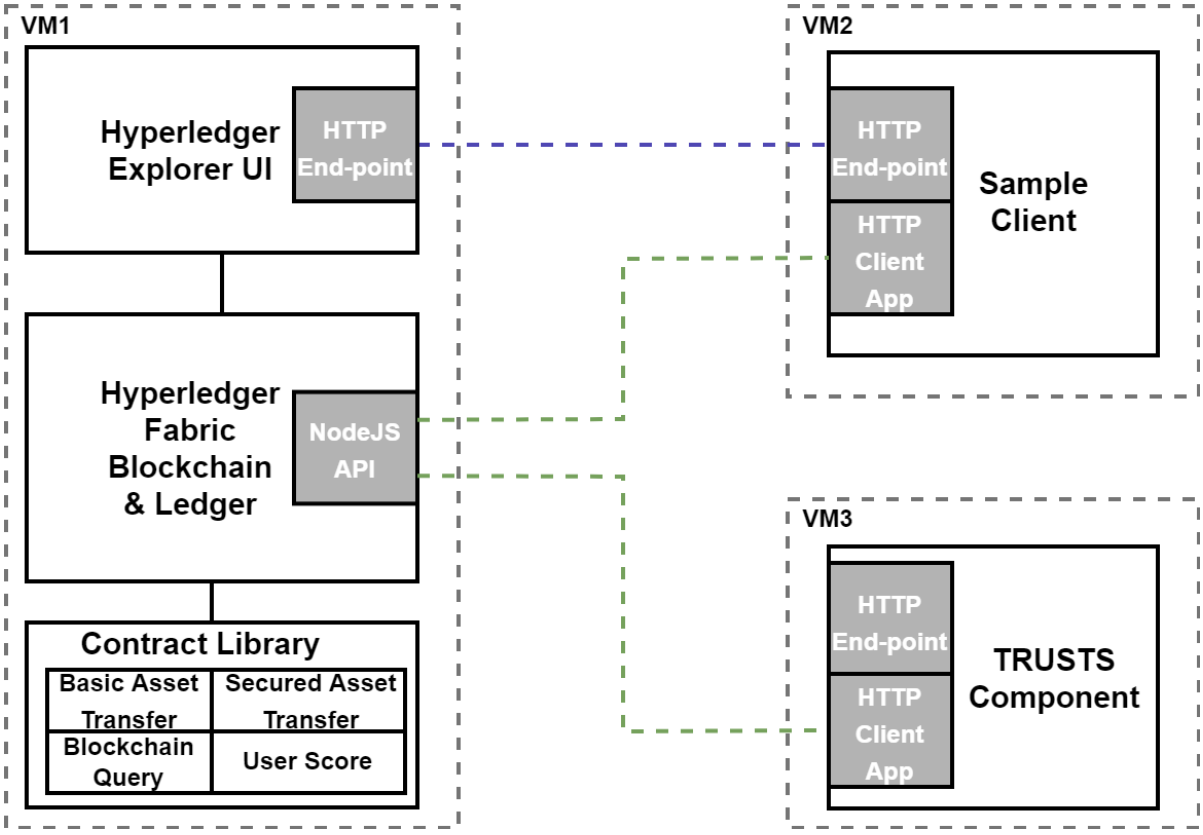

Figure 8: Smart contract demonstrator architecture.

7.1 Blockchain - Hyperledger Fabric Software

The Hyperledger fabric software enables the development of blockchain and smart contracts solutions. It is highly modular and extensible, boasts robust documentation and sample software, and has a “low barrier of entry” for setting up a blockchain network running smart contracts. For these reasons it was selected by consensus of Task 3.2 partners. The extensibility, modularity and wide breadth of supporting documents and technical material allows for partners who wish to build their own custom smart contracts or accompanying systems which were not covered in the scope of this demonstrator, to do so relatively painlessly - meaning that the demonstrator not only serves as a showcase of blockchain and smart contract functionality within the TRUSTS project, but also provides a foundation which can be further built upon.

Hyperledger Fabric provides an SDK to create a gateway on the blockchain host which we used to expose a custom API with a REST-like format, though the SDK enables the definition of one's own API so conforming to a defined standard, such as REST, is possible. A technical manual accompanies the demonstrator and provides a description of how to replicate the demonstrator installation process to set it up on one's own platform, install the Hyperledger explorer UI, and use the developed client-side API. The Hyperledger suite of tools and applications is developed by a Linux foundation hosted open-source consortium. All components of the smart contract demonstrator run as containerized applications.

7.2 Hyperledger Explorer UI

Hyperledger explorer is best described as a "blockchain visualization interface". Explorer is a web application that exposes a user interface. This UI visualizes data and processes within the blockchain to make the information easily interpretable.

7.2.1 Explorer UI Dashboard Overview

The dashboard is the front page of the Explorer UI and from here most of the core functionality is accessible. As can be seen in Figure 9, the data includes blocks, transactions, nodes and chaincodes - which is Hyperledger terminology for smart contracts. There are several tabs in the explorer UI which provide the ability to perform a granular search of various facets of the blockchain. The explorer UI excels in presenting machine-readable data on the blockchain in a human readable format. The explorer utility consists of a database, which stores blockchain information to be exposed by the UI, a web server which communicates with the blockchain to gather the data, and authorization mechanisms to control access to the explorer utility and any sensitive data it may expose.

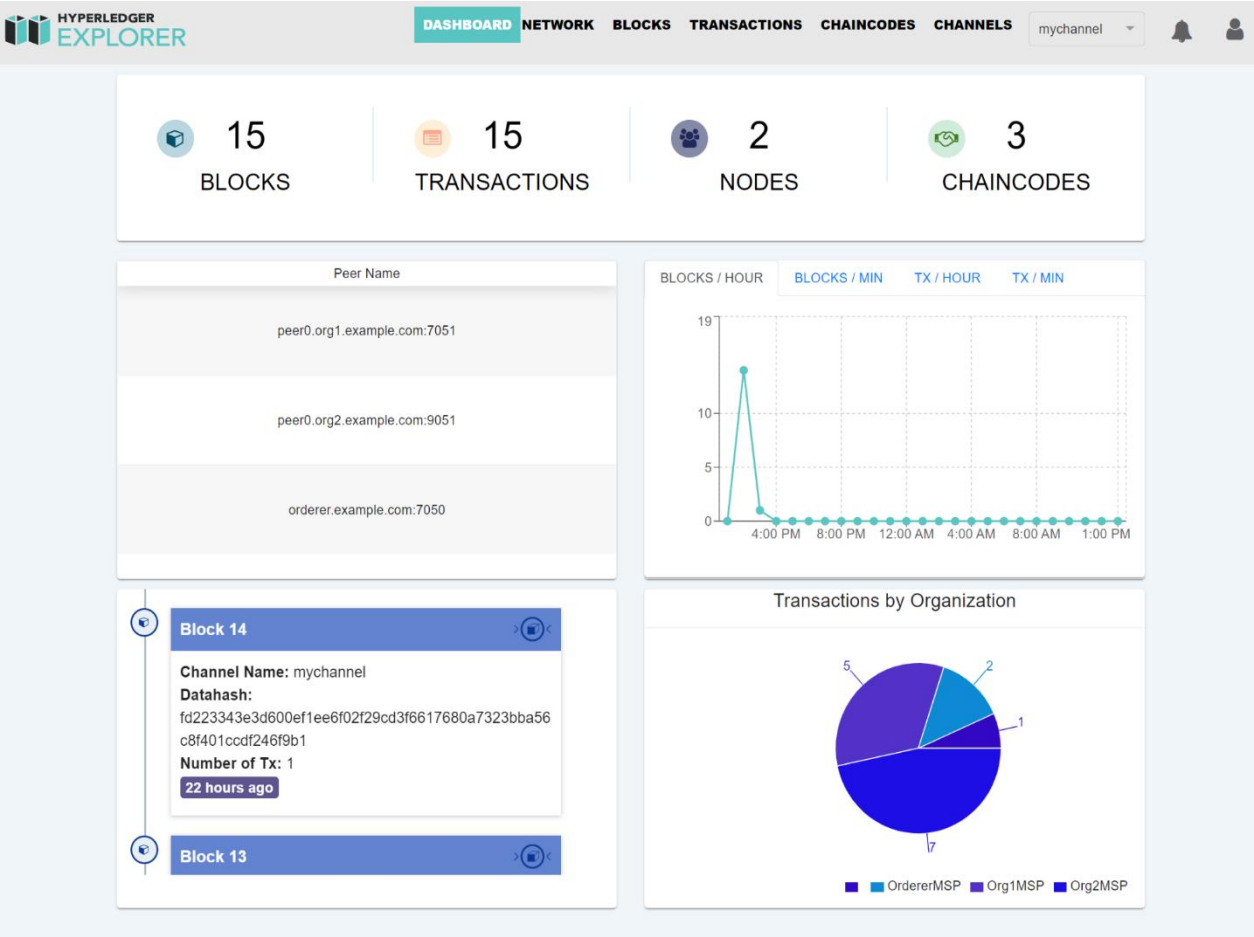

Figure 9: Hyperledger Explorer UI dashboard & sample data.

7.2.2 Explorer UI Search

A relatively standard search mechanism is included in the explorer UI which allows for blocks and transactions to be searched according to several parameters including date ranges, unique identifiers and hash codes, as well as the user or organization relevant to a particular item. Since this functionality comes packaged with the Hyperledger Explorer UI, it combines well with other blockchain search utilities such as the blockchain query smart contract, as well as the remote API's custom blockchain search. These utilities combined form a strong suite of search mechanisms.

7.3 Contract Library

The contract library holds several example (but also usable and functional) smart contracts, called "chaincodes" in Hyperledger terms. The Hyperledger fabric software package includes many sample chaincode applications which work out-of-the-box with a running Hyperledger fabric instance. Two of these Hyperledger samples were modified and underpin the demonstrator contract library, namely "asset transfer basic", "ledger queries" and "asset transfer - secured agreement".

7.3.1 Basic Asset Transfer

This chaincode is, as the name suggests, a basic example. It is based off of the Hyperledger sample "asset-transfer-basic" with some modifications of properties: removing unsuitable assets properties from the chaincode and adding necessary ones which were not present - such as a URL property, and an asset review score property. This chaincode enables the creation of assets, querying assets, the updating or addition of properties on assets, including the transfer of an asset's ownership. This chaincode does not apply any form of access control or restriction of who can edit asset properties or perform transfers; it is inappropriate for "real-world" scenarios but is useful for showcasing the demonstrator's functionality and for basic testing of components with blockchain integration and usage.

Though the chaincode is basic, and as mentioned not suitable for real-world scenarios, one scenario in which it would be appropriate is industrial rapid prototyping. Rapid prototyping occurs often within large industrial firms and is particularly pronounced in research departments of information technology companies. A rapid prototyping cycle can range from two to six weeks and may be carried out as a follow-on from a written study, or as a feasibility examination. These prototypes are built inside of a secure corporate network, and typically on a subdivision of that network such as a research lab, and the resulting prototype is not close to a product.

While such initial prototypes don't have to adhere to the stringent standards that products do, they also do not generate cash-flow directly meaning there is a reluctance to spend any more time than is necessary on such prototyping. The result of this is that security and access control are not as much of a concern as speed of development. In the context of rapid prototyping or testing a product or service against a blockchain - a simple chaincode such as the basic asset transfer could be appropriate.

The following list describes the chaincode's normal operation:

- User1 authenticates
- User1 queries the blockchain and gets a list of assets
- User1 creates an asset
- User1 changes properties on an asset
- User1 changes owner (property) of the asset
- User1 queries again and gets a list with the updated properties

7.3.2 Secured Asset Transfer

Where the basic asset transfer simply showcases the functionality of the contracts and blockchain, the secured asset transfer contract better represents a real-world transaction between two actors. This chaincode collates functionality from multiple Hyperledger fabric samples. These collated chaincodes include private data - which is a sample aimed at showcasing management and usage of private data sets in a blockchain and chaincode scenario, including use of hash codes for authenticity and validity checks of a data asset against the hash code stored on the blockchain. The state-based endorsement sample shows the endorsement policy feature of Hyperledger fabric, which specifies which peers, or users must endorse a chaincode for a given transaction to be considered valid.

The secured asset transfer chaincode is used to securely perform asset transfer between two authenticated parties, both of whom must consent to the transfer. The following is a high-level description of the typical operation of the secured asset transfer:

- User1 authenticates on their machine
- User2 authenticates on their machine
- User1 creates an asset owned by them
- User1 queries their private collection on blockchain and gets newly created asset info
- User1 queries the blockchain for ownership record, stored publicly
- User1 marks asset as ‘for sale’ at a specified price
- User2 searches the blockchain for ‘for sale’ assets and sees the asset
- User2 tries to modify the asset but cannot be due to User1 ownership
- User2 tries to buy the asset at lower than asking price
- The asset transfer fails endorsement as the users have differing prices
- The users negotiate (out of band) and come to an agreement on price
- The asset transfer succeeds and User2 can update the description

7.3.3 Review Score

The review score chaincode was developed to handle the associating of a review score with a seller or organization, meaning their trustworthiness, responsiveness or likewise. In the basic asset transfer chaincode, it was possible to simply add a property which holds the review score of an asset, but it was not possible to add the scores of sellers and reviewers to the basic asset transfer chaincode, due to the fact that the chaincode is framed around the idea of an asset, adding a property here which describes the rating of a seller or user would be out of place. Instead, the basic asset transfer chaincode was replicated, to act as a foundation for a chaincode in which the ‘asset’ is in fact a seller or organization, thereby producing a query able entry in the blockchain ledger to retrieve the particulars of a given seller or user.

This chaincode defines “Score” objects which contain two properties: a unique string ID which refers to an Organization ID, and an integer score value. Operations that can be performed on Score objects follow a similar flow to the basic asset transfer chaincode. Scores can be created, updated and deleted. Two key changes implemented in this contract are the removal of an asset owner as well as the ability to apply a more restrictive access policy, effectively enforcing which members of the blockchain can create, update and query the Score objects on the ledger. An overview of the Review Score chaincode flow:

- Admin1 Authenticates
- Admin1 Initializes the ledger with Score objects for each Organization (Org1)
- Admin1 Updates score value for organization Org1
- Admin1 Queries the ledger to return all Scores
- Org1 can view all scores but is unable to update their own score

7.3.4 Query Blockchain

In order to search or get data from the blockchain ledger requests to the blockchain are needed. The “query blockchain” chaincode is a Hyperledger fabric sample which provides this functionality. We do not use this chaincode as a standalone, instead the aforementioned chaincodes in this section leverage the “query blockchain” chaincode’s functionality behind the scenes to search the blockchain ledger and get data via requests. The chaincode supports a variety of expressive query methods, such as key/value matches, block data, value range and transaction type. The Hyperledger explorer graphical user interface is preferable to terminal-based querying from a human-readability standpoint, however in the current

implementation of the demonstrator the Explorer UI exposes public blockchain data only, and cannot expose private blockchain data, such as the particulars of a private transaction between two users, whereas one such user could authenticate in the terminal and perform a query of private data they have access to.

7.3.5 Properties List for Smart Contract Demonstrator

Figure 10, Figure 11, and Figure 12 summarize the blockchain properties of the smart contract demonstrator. These properties are chaincode-specific and are accessible over the remote API. The figures in this section showcase the property names on the left along with an example of the kind of values that may be stored in the property on the right. This properties list is extensible with relatively little effort and two rounds of communication with project partners were carried out to define this list, including the Task 3.3 partners regarding interoperability.


```
"ID": "asset11",
"title": "Large medical dataset",
"transactionId":"4eb70ebaf964cf049a9803fbd8c9f1b5050d2de3d7c6aadd74e94841129bda4c",
"size": 5,
"owner": "Org1MSP",
"appraisedValue": 15,
"publisher": "TRUSTS",
"creator": "TRUSTS_RESEARCH",
"contactPoint": "user.trusts@trusts.eu",
"keyword": "Medical",
"Authorisation": "OAuth",
"Dataaccess": "URL:www.example.com/data",
"creationdate": "07/02/22",
"license": "MIT",
"format": ".csv:none:csv",
"accessInterface": "Web:Local",
"description": "A sample asset on the trusts platform"
```


Figure 10: Basic asset transfer blockchain properties with sample values.

```
"object_type": "asset_properties",
"assetid": "asset1",
"title": "Large medical dataset",
"size": 5,
"Publisher": "TRUSTS",
"Creator": "TRUSTS_RESEARCH",
"ContactPoint": "Joe Trusts",
"Keyword": "Medical",
"Authorisation": "OAuth",
"DataAccess": "URL:www.example.com/data",
"CreationDate": "07/02/22",
"License": "MIT",
"Format": ".csv:none:csv",
"AccessInterface": "Web:Local",
"Description": "A sample asset on the trusts platform",
"salt": "3932"
```


Figure 11: Secured asset transfer blockchain properties with sample values.


```
"ID": "d72b8741-0326-4efd-8618-ce6fc0b2047d",
"score": 10,
"target": "Asset1",
"author": "Org2_appuser",
"description":"A goodasset on the trusts platform",
```


Figure 12: User scoring blockchain properties with sample values.

7.4 NodeJS API

The blockchain remote API component is built on Hyperledger Fabric's official SDK for developing client applications and was developed during this task to enable remote connections to the blockchain, along with some important auxiliary functionalities such as granular search. The SDK provides the foundation for blockchain interaction with a range of integrated tools and languages. Within the context of the demonstrator, the blockchain API was developed using the Hyperledger Node SDK. The Node SDK provides a JavaScript API for Hyperledger functionality, including user access control with Hyperledger Wallets, gateway and networking abstraction and finally, the execution of a chaincode within the blockchain.

7.4.1 User Accounts & Access Control

A basic form of access control is implemented in the API using the Hyperledger SDK wallet. Hyperledger Fabric is a permissioned blockchain, meaning all users must be identified and authenticated before being permitted to perform actions on the blockchain. Wallets are the mechanism used to implement this functionality in a Fabric blockchain.

A Wallet contains identities that can be referenced by the user of an application. These IDs can be issued by a certificate authority or generated internally. When connecting to the API the user must provide their ID and Membership Service Provider, which are used to identify and authorize a user and generate a Connection Profile (in-memory object containing network configuration).

There are three types of wallets in Hyperledger, defined by the storage mechanism: in-memory, filesystem and database. The API implements two file-system wallets, one for each Organization in the demonstrator blockchain, each wallet defines IDs for an organization's admin and user accounts. Additional organizations can be added to the blockchain, as well as additional users which are defined as the API application level.

7.4.2 Fabric Gateway

Each Peer in a Hyperledger blockchain communicates through gRPC and exposes a Gateway component, this component simplifies and abstracts the transactional and networking logic from blockchain operations. Client applications deployed do not directly communicate with network peers, but rather implement a Gateway interface provided in the Node SDK.

The Gateway connection is established in the API by passing a user requests Connection Profile, Organization ID, and Wallet. Gateway discovery takes place on the localhost in the current architecture but can be configured to externalize the API from the blockchain host.

7.4.3 Chaincode Execution

The NodeJS API defines a total of seventeen (17) HTTP endpoints across three chaincodes deployed to the blockchain. Endpoints URLs follow the format:

http:// machine-IP:port/<chaincode>/<function >

Each endpoint maps one-to-one to a chaincode function call and accepts incoming client HTTP requests. The request body contains varying parameters, depending on the specific function, there are optional parameters that must be provided. There are three (3) required parameters in all HTTP requests to the API:

- Membership Service Provider
- User ID
- Channel ID

These parameters are used to identify and authorize the users request before any further operations. Further optional parameters are specific to the function being called, but may include asset properties, trade parameters or score parameters.

In the event of unauthorized access, chaincode errors or user errors, a basic error handler is implemented to return the relevant information to the client application, a response object is returned to the client application containing a status code and message.

7.4.4 Blockchain Search

Included in the NodeJS API is a blockchain search utility which allows for a granular, multi-faceted search to be applied over the blockchain in order to produce a set of results. This blockchain search functionality is provided by the remote API module in addition to the existing Hyperledger Fabric “query blockchain” baked-in functionality. The search terms are set up as key value pairs consisting of the term type, including “owner, category, license” and others, as well as the specific field pertaining to this category such as: “Mark Griffith, Medical, GPLv2”. The blockchain entries which match the criteria of the search will be returned, and of course how specific this criterion is typically influences the number of blocks returned.

7.4.5 Transaction ID Return and Explorer UI Link

The ID of a specific transaction is an attribute which can be returned through the NodeJS API. This attribute is unique to a specific transaction and is of particular importance as it can be used to form a direct URL link to an Explorer UI representation of the blockchain entry (see Figure 13). This entry contains a multitude of data about the specific transaction, some of the more interesting articles of which are highlighted in Figure with a red arrow. These articles are in fact data points related to the underlying asset represented by this blockchain entry.

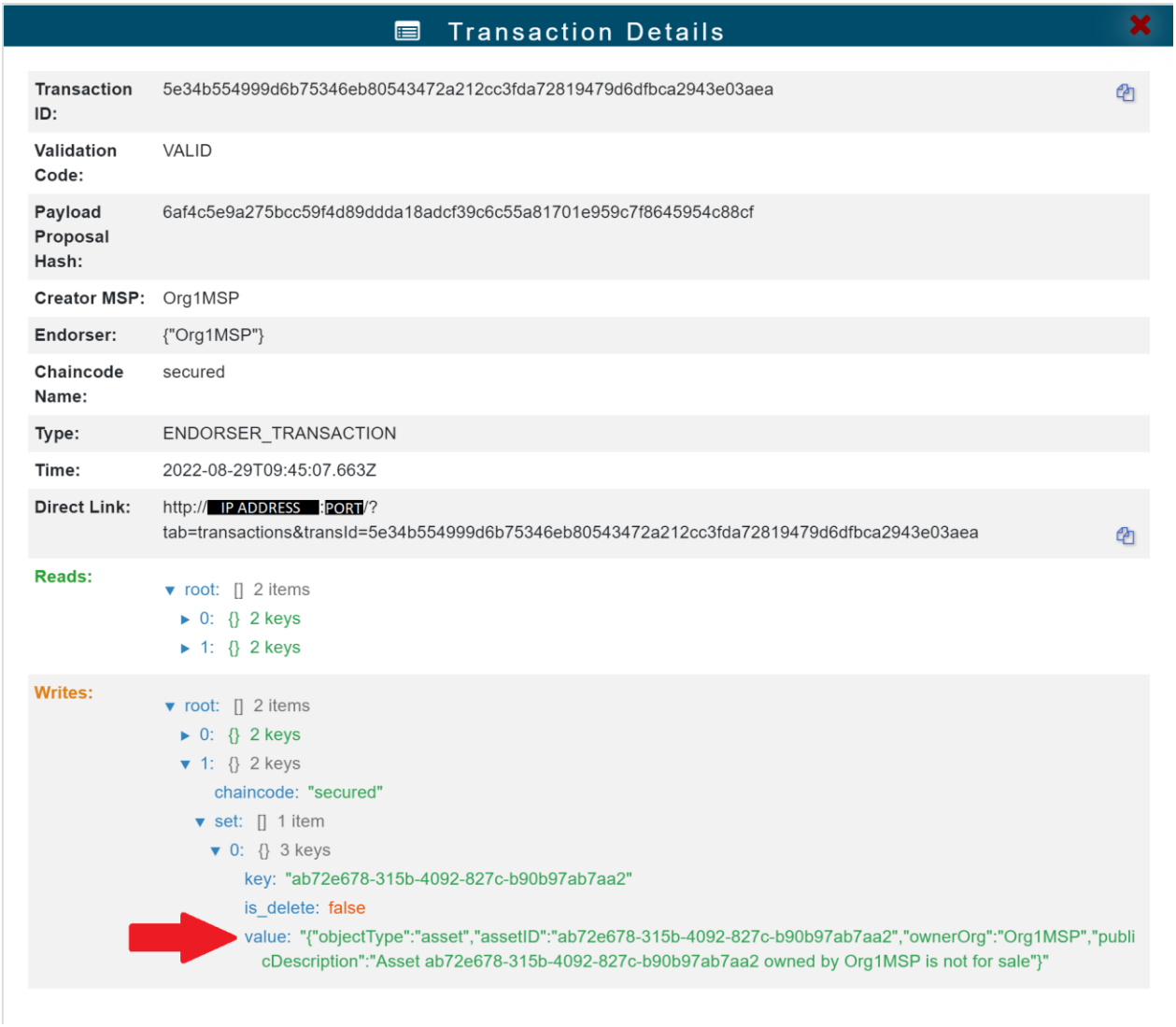

Figure 13: Hyperledger Explorer UI representation of blockchain entry.

7.4.6 Demonstrator Client

A collection of sample HTTP requests are deployed in the demonstrator platform in order to emulate client requests to the API. These sample requests include necessary request data, format and endpoints to test the API and each of the chaincodes. Combined with the API specification, this provides a rapid deployment and testing environment of the API functionality.

7.5 Payment Compatibility Demonstrator

This module was developed on top of the remote API blockchain search functionality. The search is used to find an asset by the asset ID, and the resulting asset information is taken from the retrieved results and displayed on the Web UI. This module was developed using JavaScript, NodeJS and the Express Framework,

as well as the Hyperledger SDK to make requests to the blockchain. See Figure 14 which shows the web UI. A payment system is then selected, for this compatibility demonstrator we chose PayPal, Google Pay and Revolut.

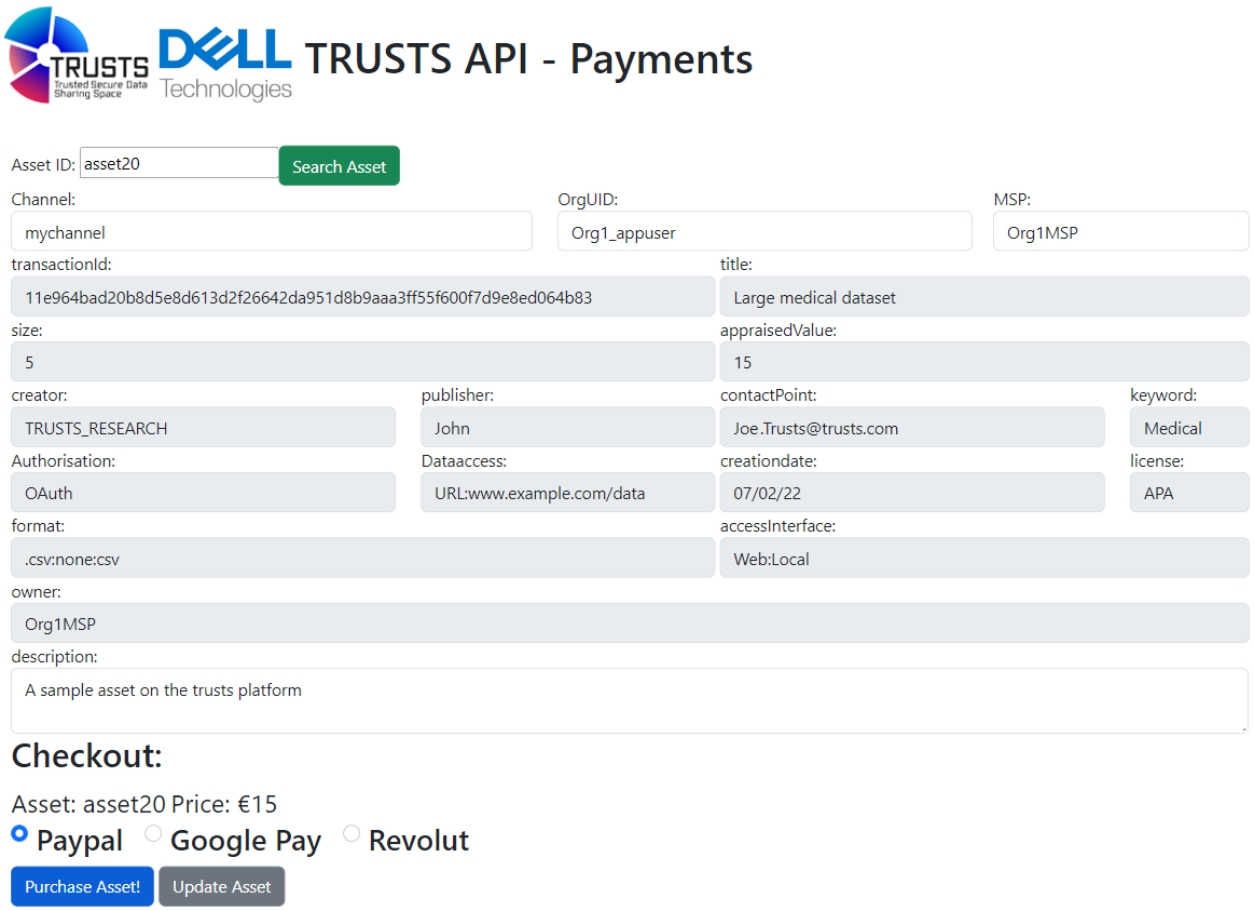

Figure 14: Web UI of payment system compatibility demonstrator.

Once the purchase asset option is selected the user will be taken to the relevant payment system populated with the email address given in the contact point field (see Figure 15). Once there the user will complete the purchase out-of-band using the selected payment system. In this scenario once the purchase has been made the description box can be updated with the payment UUID. Many of the manual elements of this process could be removed via a deeper level of integration with a payment system provider's SDK.

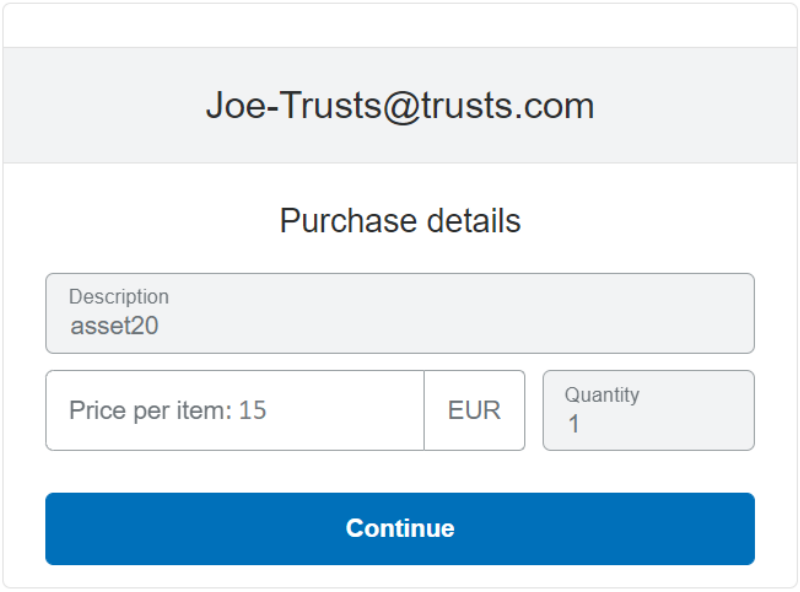

Figure 15: Out-Of-Band payment system order page example.

Once the payment is complete an invoice similar to the example invoice in the below figure will be returned from the out-of-band payment system to the user, and in this invoice is typically located some unique code to identify the specific payment. Payment platforms such as those mentioned in this section often include dispute resolution systems, and other features which gives the TRUSTS platform the option to outsource these important items, see the bottom of Figure 16 to see some typical notices included in invoices which describe these kinds of systems.

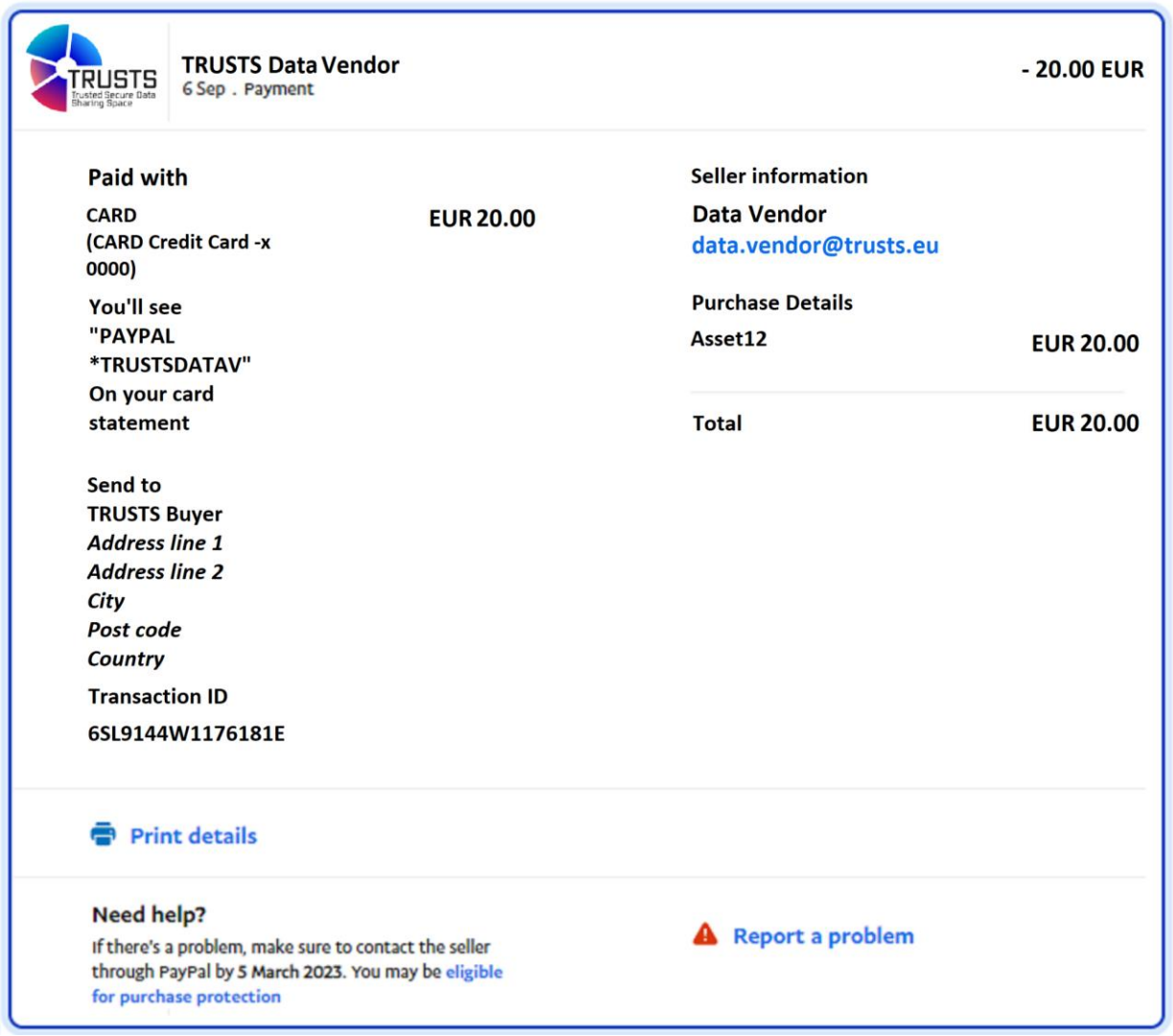

Figure 16: Invoice example.

8 Conclusions and Next Actions

The outcome of the project contains the necessary concepts for the deployment of smart contracts by means of a smart contract component in the context of the European data market, provided by TRUSTS from both a technical and a legal perspective. At the beginning, an overview of the basic information of the technologies underlying the smart contract component was provided. In addition to the definition of a blockchain and smart contracts, the IDS and the DMA, which are the underlying technologies of TRUSTS, were also described in more detail. A general legal overview of smart contracts, the legal challenges of applying contract law to smart contracts, and how the entire topic can be applied in accordance with EU law were then examined. The subsequent technical section describes the technical challenges of implementing smart contracts in the European data market TRUSTS. In addition to the functional and technical requirements for a blockchain application with smart contracts execution, it was explained how these requirements can be met. Furthermore, the example of a sequence diagram was used to illustrate how such a blockchain application can be connected in TRUSTS. Finally, an overview was given of various problems, risks and disadvantages in the use of blockchains and smart contracts that must be taken into account in the technical implementation of a blockchain application with smart contracts execution. Section 6 describes an overview of methods and potential issues that arise in the privacy and security of smart contracts. Various techniques for maintaining a high level of data protection and security have been explained, and vulnerabilities have been identified, their causes explained, and how they can be addressed. The concept concludes with the presentation of an example implementation of a smart contract demonstrator. Section 7 describes such a demonstrator based on all the previous sections mentioned, which, in addition to a pure blockchain with smart contract execution, has a user interface, a library of different smart contracts and its own API. The latter was also used to demonstrate payment compatibility, i.e., the smart contract demonstrator can theoretically be used for payment transactions as well.

The contribution of Task 3.2 is consistent with the goals of the overall project, as TRUSTS aims to provide various components for a secure European data market and Task 3.2 describes the concepts for one of these components. A data market thrives on transactions between participants. However, in order to be able to quickly clarify ambiguities, for example, a component is needed within the data market that can provide a legally and technically secure overview of all transactions in the market. In this component, all transactions that have occurred in the past between participants are stored in a blockchain. In this way, manipulation can be prevented, which in turn increases security in the data market and trust between the participants.

The concept created brings several benefits to the overall project for future development and use beyond the life of the project. For example, there is the economic impact. The deliverable represents how a blockchain component with smart contract execution can look like. It also discusses quite a few technical and legal challenges that need to be considered during implementation. This makes it possible to increase security when implementing such a component for a data market. As a result, there is increased trust among participants, and they are more likely to participate in the TRUSTS data market. Further, the blockchain component offers the possibility of providing interfaces to payment service providers in future developments, as illustrated in Section 7. Through this, payments could be automated between participants when the smart contracts verify a transaction.

9 References

[1] Stahl, F., Schomm, F., Vossen, G. (2016). A Classification Framework for Data Marketplaces. Vietnam Journal of Computer Science 3, 137–143. https://doi.org/10.1007/s40595-016-0064-2

[2] Otto, B., Steinbuß, S, Teuscher, A. & Lohmann, S. (2019). IDS Reference Architecture Model, v3.0. International Data Spaces Association. https://internationaldataspaces.org/use/reference-architecture/

[3] W. Agahari, M. d. Reuver, and T. Fiebig. D. U. o. Technology. (2019). Understanding how privacy-preserving technologies transform data marketplace platforms and ecosystems: The case of multi-party computation

[4] P. Sharma, S. Lawrenz, and A. Rausch, "Towards Trustworthy and Independent Data Marketplaces," 2020 2020: ACM, doi: https://doi.org/10.1145/3390566.3391687

[5] P. Koutroumpis, A. Leiponen, and L. D. W. Thomas, "Markets for data," Industrial and Corporate Change, vol. 29, no. 3, pp. 645-660, 2020, doi: https://doi.org/10.1093/icc/dtaa002

[6] M. Fruhwirth, M. Rachinger, and E. Prlja, "Discovering Business Models of Data Marketplaces," in Proceedings of the 53rd Hawaii International Conference on System Sciences, 2020

[7] Höchtl, J., & Lampoltshammer, T. J. (2017, May). Social implications of a data market. In Conference for E-Democracy and Open Government (p. 177).

[8] Virkar, S., Viale Pereira, G., & Vignoli, M. (2019, September). Investigating the Social, Political, Economic and Cultural Implications of Data Trading. In International Conference on Electronic Government (pp. 215-229). Springer, Cham.

[9] Christidis, K.; Devetsikiotis, M. (2016): Blockchains and Smart Contracts for the Internet of Things. In: IEEE Access. 4, S. 2292-2303, IEEE Xplore Document. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7467408

[10] Prinz W.; Schulte, A. (2017): Blockchain und Smart Contracts Technologien, Forschungsfragen und Anwendungen. Fraunhofer. https://www.iml.fraunhofer.de/content/dam/iml/de/documents/OE260/Fraunhofer-Positionspapier_Blockchain-und-Smart-Contracts.pdf

[11] Burelli, F.; John, M.; Cenci, E.; Otten, J. Courtneidge, R.; Clarence-Smith, C. (2015): Blockchain and Financial Services: Industry Snapshot and Possible Future Developments. URL: https://www.innovalue.de/publikationen/InnovalueLockeLord-BlockchaininFinancialServices2015.pdf

[12] Nakamoto, S. (2008): Bitcoin: A Peer-to-Peer Electronic Cash System. URL: https://bitcoin.org/bitcoin.pdf

[13] Greenspan, G. (2015): MultiChain Private Blockchain – White Paper. URL: https://www.multichain.com/download/MultiChain-White-Paper.pdf

[14] Swan, M. (2015): Blockchain - Blueprint for a New Economy. O'Reilly Media, Inc. Februar 2015.

[15] Triangle. 2017. Maersk and IBM launch blockchain-based cross-border supply chain solution. 2017. http://postandparcel.info/78528/news/maersk-and

[16] Smith, J. 2016. Blockfreight™ Whitepaper v1.0.1. 2016

[17] Skuchain. 2017. Skuchain Brackets - Blockchain Technology for Collaborative Commerce. 2017

[18] Popper, N. und Lohr, S. 2017. Blockchain: A Better Way to Track Pork Chops, Bonds, Bad Peanut Butter? 2017. https://mobile.nytimes.com/2017/03/04/business/dealbook/blockchain-ibm-bitcoin.html

[19] IBM. 2017. Maersk and IBM Unveil First Industry-Wide Cross-Border Supply Chain Solution on Blockchain. 2017. http://www-03.ibm.com/press/us/en/pressrelease/51712.wss

[20] Everledger. 2017. Everledger - A Digital Global Ledger. 2017

[21] Szabo, N. (1994): Smart Contracts. URL: http://web.archive.org/web/20160306112751/http://szabo.best.vwh.net/smart.contracts.html

[22] Szabo, N. (1996): Smart Contracts: Building Blocks for Digital Markets. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html

[23] Chen, S.; Shi, R.; Ren, Z; Yan, J.; Shi, Y; Zhang J. (2017): A Blockchain-based Supply Chain Quality Management Framework. IEEE Computer Society. DOI 10.1109/ICEBE.2017.34. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8119146

[24] Y. Hanada, L. Hsiao and P. Levis, "Smart Contracts for Machine-to-Machine Communication: Possibilities and Limitations," 2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS), 2018, pp. 130-136, doi: 10.1109/IOTAIS.2018.8600854. URL: https://ieeexplore.ieee.org/document/8600854

[25] Wood, Daniel Davis. “Ethereum: A secure decentralized generalized transaction ledger.” (2014). URL: https://gavwood.com/paper.pdf

[26] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., ... & Yellick, J. (2018, April). Hyperledger fabric: a distributed operating system for permissioned blockchains. In Proceedings of the thirteenth EuroSys conference (pp. 1-15). URL: https://arxiv.org/pdf/1801.10228

[27] Hepp, T., Sharinghousen, M., Ehret, P., Schoenhals, A. & Gipp, B. (2018). On-chain vs. off-chain storage for supply- and blockchain integration. it - Information Technology, 60(5-6), 283-291. https://doi.org/10.1515/itit-2018-0019

[28] C. Li, B. Palanisamy and R. Xu, "Scalable and Privacy-Preserving Design of On/Off-Chain Smart Contracts," 2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW), 2019, pp. 7-12, doi: 10.1109/ICDEW.2019.00-43

[29] Chen, S.; Shi, R.; Ren, Z; Yan, J.; Shi, Y; Zhang J. (2017): A Blockchain-based Supply Chain Quality Management Framework. IEEE Computer Society. DOI 10.1109/ICEBE.2017.34. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8119146

[30] ww.ibm.com. (2021). Security Bulletin: IBM MQ Blockchain bridge dependencies are vulnerable to an issue in Apache Log4j (CVE-2021-44228). [online] Available at:

https://www.ibm.com/support/pages/security-bulletin-ibm-mq-blockchain-bridge-dependencies-are-vulnerable-issue-apache-log4j-cve-2021-44228 [Accessed 17 Jan. 2022].

[31] Greig, J. (n.d.). Apache releases new 2.17.0 patch for Log4j to solve denial of service vulnerability. [online] ZDNet. Available at: https://www.zdnet.com/article/apache-releases-new-2-17-0-patch-for-log4j-to-solve-denial-of-service-vulnerability/ [Accessed 17 Jan. 2022].

[32] Neubauer, M. Goebel, A. (2018). Blockchain for Off-Chain Smart Contracts in an SAP® Environment https://infohub.delltechnologies.com/static/media/9e19284a-31f5-4f98-b716-45447f8a9f11.pdf [Accessed 23 September 2022]

[33] Solomon, R. Almashaqbeh, G. (2021). smartFHE: Privacy-Preserving Smart Contracts from Fully Homomorphic Encryption. Access: https://eprint.iacr.org/2021/133.pdf [Accessed 23 September 2022].

[34] Ren, Q. Liu, H. Li, Y. Victor, A. Lei, H. Wang, L. Chen, B. (2022). CLOAK: Enabling Confidential Smart Contract with Multi-party Transactions. https://arxiv.org/pdf/2106.13926.pdf [Accessed 23 September 2022]

[35] Huixin Wu, Feng Wang, "A Survey of Noninteractive Zero Knowledge Proof System and Its Applications", The Scientific World Journal, vol. 2014, Article ID 560484, 7 pages, 2014. https://doi.org/10.1155/2014/560484

[36] Anderson, R. (2003). “Trusted computing” Frequently asked questions.

[37] The Smart Contract Security Mindset https://consensys.net/blog/developers/the-smart-contract-security-mindset/ [Accessed 03 October 2022]

[38] Nguyen, (2019). Impact of network delays on Hyperledger Fabric. Access: https://hal.archives-ouvertes.fr/hal-02073764/document [Accessed 03 October 2022]

[39] NIST National Vulnerability Database (2022). Access: NVD - CVE-2022-35253 (nist.gov) [Accessed 03 October 2022]

[40] Smart Contract Security: What Are the Weak Spots of Ethereum, EOS, and NEO Networks? https://hal.archives-ouvertes.fr/hal-02073764/document [Accessed 03 October 2022]

[41] Kockan C, Zhu K, Dokmai N, Karpov N, Kulekci MO, Woodruff DP, Sahinalp SC. Sketching algorithms for genomic data analysis and querying in a secure enclave. Nat Methods. 2020 Mar;17(3):295-301. doi: 10.1038/s41592-020-0761-8. Epub 2020 Mar 4. PMID: 32132732; PMCID: PMC7423249.

[42] K. N. Mallikarjunan, K. Muthupriya and S. M. Shalinie, "A survey of distributed denial of service attack," 2016 10th International Conference on Intelligent Systems and Control (ISCO), 2016, pp. 1-6, doi: 10.1109/ISCO.2016.7727096.

[43] Biswas, Saikat & Sohel, M. & Sajal, Md.Mizanur & Afrin, Tanjina & Bhuiyan, Touhid & Hassan, Md Maruf. (2018). A Study on Remote Code Execution Vulnerability in Web Applications.

[44] Shi, Gang & Ru, Jiangtao. (2016). Research on Classification of Memory Attack. 10.2991/wartia-16.2016.78.

[45] Raskin M., The law and legality of smart contracts, Law Technology Review 304 (2017), Georgetown (USA).

[46] Savelyev A., Contract law 2.0: 'Smart' contracts as the beginning of the end of classic contract law, Information & Communications Technology Law, 26:2 (2017), DOI: 10.1080/13600834.2017.1301036

[47] Sklaroff J., Smart Contracts and the Cost of Inflexibility, University of Pennsylvania Law Review, Vol. 166:263.

[48] Lessig L., Code and other laws of the cyberspace, Basic Book (2012).

[49] European Parliament, Committee for Legal Affairs, Draft Report with recommendations to the Commission on a Digital Services Act: adapting commercial and civil law rules for commercial entities operating online, 2020.

[50] Kasatkina M., The Interpretation of Smart Contracts in the EU and the US, International Comparative Jurisprudence 2021 Volume 7 Issue 2.

[51] Cannarsa M., Interpretation of Contracts and Smart Contracts: Smart Interpretation or Interpretation of Smart Contracts?, European Review of Private Law 6-2019 [773-786], 2019.

[52] Ferreira A., Regulating smart contracts: Legal revolution or simply evolution?, Telecommunication Policy 45 (2021)

[53] Novoselova L. A., Tokenization of objects of civil law. Economy and Law, 12, 29–44.

[54] Ghodoosi F., Contracting in the Age of Smart Contracts, Washington Law Review, 96:51, 2020.

[55] International Swaps and Derivatives Association (ISDA) & Linklaters, Whitepaper: Smart Contracts and Distributed Ledger – A Legal Perspective, 2017.

[56] Drummer D., Neumann D., Is code law? Current legal and technical adoption issues and remedies for blockchain-enabled smart contracts, Journal of Information Technology 2020, Vol. 35(4) 337–360, pp. 340-341.

[57] Alpa G., European Private Law, Conceptions and Definitions of Contract, European Business Law Review Volume 27, Issue 6 (2016) pp. 709 – 718.

[58] German Civil Code (BGB), Division 3, Title 3, Sections 145-157.

[59] French Civil Code, Articles 1113-1122.

[60] Italian Civil Code, Articles 1326-1335.

[61] Stover v Manchester City Council [1974] 1 WLR 1403; Moran v University College Salford (No 2), The Times, November 23, 1993.

[62] Wéry P., Droit des obligations, Brussels, Larcier (2011).

[63] Von Bar, C., Clive, E., & Schulte-Nölke, H. (2009). Principles, definitions and model rules of European private law: Draft Common Frame of Reference (DCFR). Walter de Gruyter.

[64] See analysis of English law on this issue by Hepple B. A., Intention to create legal relations, in Cambridge Law Journal 28 (1970)

[65] European Commission, Directorate-General for Communications Networks, Content and Technology, Directorate F — Digital Transformation, Unit F3 — Digital Innovation and Blockchain, and Schrepel T., Smart Contracts and the Digital Single Market Through the Lens of a "Law + Technology" Approach, 2021, p. 33.

[66] Kolber A. J., Not-So-Smart Blockchain Contracts and Artificial Responsibility, 21 Stanford Technology and Law Review, 198 (2018)

[67] UK 1967 Misrepresentation Act.

[68] Italian Civil Code, Articles 1439-1440

[69] French Civil Code, Article 1116.

[70] Italian Civil Code, Articles 1434-1436

[71] French Civil Code, Articles 1111-1115.

[72] Directive 2011/83/EU of the European Parliament and of the Council of 25 October 2011 on consumer rights, amending Council Directive 93/13/EEC and Directive 1999/44/EC of the European Parliament and of the Council and repealing Council Directive 85/577/EEC and Directive 97/7/EC of the European Parliament and of the Council.

[73] Werbach K., Cornell N., Contracts ex machina, Duke law journal, 2017, Vol. 67 (2), p. 340.

[74] European Commission, Schrepel T.

[75] Zheng Z. et al., An overview on smart contracts: Challenges, advances and platforms, Future Generation Computer Systems 105 (2020) 475–491.

[76] The UK Law Commission (cited supra, note 23) also appears to recognize that, to the extent that the technological complexity of smart contracts is explained and made understandable, smart contracts can be conducive to greater legal certainty than traditional contracts.

[77] Koulu R., Blockchains and Online Dispute Resolution: Smart Contracts as an Alternative to Enforcement, HeinOnline, Volume 13, Issue 1, May 2016

[78] European Commission

[79] Grimmelmann J., All smart contracts are ambiguous, Journal of Law and Innovation, Vol. 2, No. 1, 2017

[80] French Civil Code, Article 1135

[81] Tjong Tjin Tai E., Force majeure and excuses in smart contracts, European Review of Private Law, 6, 787–904, 2018.

[82] Gilcrest J., Carvalho A., Smart Contracts: Legal Considerations, 2018 IEEE International Conference on Big Data

[83] Article 5(1)(f) GDPR

[84] Lai, R., & Chuen, D. L. K. (2018). Blockchain–from public to private. In Handbook of Blockchain, Digital Finance, and Inclusion, Volume 2 (pp. 145-177). Academic Press

[85] Pettenpohl, H. & Spiekermann, M. & Both, J. (2022). International Data Spaces in a Nutshell. 10.1007/978-3-030-93975-5_3

[86] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo and T. Chen, "Defining Smart Contract Defects on Ethereum," in IEEE Transactions on Software Engineering, vol. 48, no. 1, pp. 327-345, 1 Jan. 2022, doi: 10.1109/TSE.2020.2989002

[87] Sillaber, C., Waltl, B. Life Cycle of Smart Contracts in Blockchain Ecosystems. Datenschutz Datensich 41, 497–500 (2017). https://doi.org/10.1007/s11623-017-0819-7

[88] Baliga, A. (2017). Understanding blockchain consensus models. Persistent, 4(1), 14