Modelling and Validating Access Control Rules based on Verifiable Credentials: Enhancing Data Privacy on the Semantic Web

Bachelor Thesis

by

Valentin Johannes Betz

Degree Course: Industrial Engineering and Management B.Sc.

Matriculation Number: 2258679

Institute of Applied Informatics and Formal Description Methods (AIFB)

KIT Department of Economics and Management

Advisor:	Dr.-Ing. Tobias Käfer
Second Advisor:	Prof. Dr. Andreas Oberweis
Supervisor:	M.Sc. Christoph Braun
Submitted:	August 1, 2023

Abstract

This thesis introduces a semantic access control system that imposes constraints on a Verifiable Credential by utilizing Web Access Control extended by a SHACL shape. This system enables the definition of required identity proofs a user must present to gain access to a Solid Pod. The data model governing this access rule is outlined in detail.
Furthermore, we introduce the Verifiable Consent Protocol, which guarantees secure and transparent data processing between a server and a client. Through the creation, exchange, and signature of usage policies, both client and server reach an agreement regarding the utilization of server resources and the processing of the client's personal data contained within the Verifiable Credential. These policies adhere to the ODRL and OAC profile standards, ensuring compliance with GDPR regulations in the processing of personal data. The data model of the data usage policies is also presented in detail.
To assess the secrurity of the protocol against dishonest agents and attackers, formal verification is conducted using ProVerif, demonstrating its secrecy and reliability. Formal verification also proves a non-repudiable agreement between the two parties.

Contents

List of Abbreviations

ACL Access Control List.

ACP Access Control Policy Language.

ACR Access Control Rule.

DPV Data Privacy Vocabulary.

GDPR General Data Protection Regulation.

HTTP Hypertext Transfer Protocol.

ODRL Open Digital Rights Language.

RDF Resource Description Framework.

URI Uniform Resource Identifier.

URL Uniform Resource Location.

URN Uniform Resource Name.

VC Verifiable Credential.

VC Model Verifiable Credential Model v1.1.

VP Verifiable Presentation.

WAC Web Access Control.

List of Figures

List of Tables

Listings

List of Namespaces

Prefix	**Namespace**
acl:	http://www.w3.org/ns/auth/acl#
acp:	http://www.w3.org/ns/solid/acp#
cred:	https://w3.org/2018/credentials#
dc:	http://purl.org/dc/terms/
dpv:	http://www.w3.org/ns/dpv#
ex:	http://example.org/
foaf:	http://xmlns.com/foaf/0.1/
oac:	https://w3id.org/oac#
odrl:	http://www.w3.org/ns/odrl/2/
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
schema:	http://schema.org/
sec:	https://w3id.org/security#
sh:	http://www.w3.org/ns/shacl#
xsd:	http://www.w3.org/2001/XMLSchema#

Table 1: Namespaces used in the thesis

The prefix definitions in code excerpts in this thesis are omitted. Prefixes are defined according to Table 1.

1 Introduction

Managing and delivering digital assets is becoming more important in a increasingly connected and data-driven world. Effective control over access to user's own information, especially personal data, is critical to protect users' privacy and data sovereignty. In this context, various approaches and technologies have evolved to define and implement access control mechanisms. We envision a system in which one can identify oneself with a digital credential in order to access a resource without the need for third party authentication. The processing of personal data from the digital credential should be transparently communicated in advance and then verifiably accepted by all parties.

The subsequent research questions follow immediately:
(a) How can requirements for authentication and authorization be modelled in a way that is interoperable with existing approaches to model Verifiable Credentials, and
(b) how can all parties be aware of all terms related to the processing of (personal) data and be able to prove a reached agreement?

This bachelor thesis deals with a comprehensive overview of important decentralized technologies and data models for access control and sharing of digital resources. The aim of the thesis is to investigate the functionality and benefits of these technologies and to unify these technologies to define data models for access rules and usage policies. Furthermore, this thesis presents a data usage agreement protocol that enables transparent and secure processing of personal data. The protocol prescribes policies that regulate the use of resources and verifiable credentials.

In particular, the following content is the focus of the thesis:

The preliminaries chapter 2 introduces several basic concepts relevant to understanding access control. These include Uniform Resource Identifier (URI), Hypertext Transfer Protocol (HTTP), Resource Description Framework (RDF), the Solid Project, the Verifiable Credential Model v1.1, the Open Digital Rights Language (ODRL) and ODRL Profile for Access Control (OAC).
Then, in Chapter 3, related work is presented that also deals with access controls, usage regulations, Verifiable Credentials and Self-Soverign Identity. The discussion of the research results enables a better understanding of the need and benefits of the technologies presented in the thesis.
In chapter 4, data models for access control rules and data usage policies are presented. The The process of developing these data models include an analysis of access control languages such as Web Access Control (WAC) and Access Control Policy Language (ACP), and constraint languages such as SHACL and ShEx. The conformity of data models with the General Data Protection Regulation (GDPR) is also considered.
Chapter 5 is dedicated to the Verifiable Consent Protocol and the use of the presented

data models for data usage policies and personal data usage policies. The focus is on how Verifiable Consent can be used as a mechanism for obtaining and proving consent for data processing.
Finally, in chapter 6, the technologies and models presented in the thesis are evaluated using formal verification procedures utilizing ProVerif. Aspects such as secrecy, reliability and non-repudiation are considered in more detail to assess the suitability and security of the Verifiable Consent Protocol presented.
By bringing together the mentioned contents and critically analysing the presented technologies, the bachelor thesis will provide a comprehensive insight into the current developments and challenges in the field of access control based on Verifiable Credentials. Furthermore, application examples and recommendations for the implementation of access control rules and data usage policies will be given.

2 Preliminaries

This chapter provides information on basic knowledge that is necessary for this thesis. The first section 2.1 is about the principle of URIs (Uniform Resource Identifier). The following section 2.2 gives an overview about the Hypertext Transfer Protocol (HTTP). After, the Resource Description Framework (RDF) is explained in section 2.3. The Solid Project is introduced in section 2.4. After, section 2 explains the Verifiable Credential Model. Then section 2.6 introduces the Open Digital Rights Language (ODRL). Finally, section 2.7 is about ODRL profile for Access Control (OAC) and DPV.

In this thesis the term 'resource' is used with the following definition: A resource is an abstract concept that describes various things that can be defined and named on the web - e.g. files, web pages, directories, agents, concepts, etc. These things do not need to be physical or digital items, they may be abstract concepts. [4]

2.1 Uniform Resource Identifier (URI)

A Uniform Resource Identifier (URI) is a limited set of characters to identify a resource [4]. A URI should adhere to the generic URI syntax from *Berners-Lee et al.* and may adhere to explicit URI schemes for certain use cases. To ensure that a URI is interpreted correctly, each URI has to begin with a scheme name. URIs from the HTTP scheme are recognizable by their 'http' prefix for example *http://example.org/public/file.txt*. Other examples are email addresses *mailto:example@kit.edu*, file paths *file:///C:/Documents/example.txt* or telephone numbers *tel:+491234567890*. Behind the scheme name and a colon, a path follows to describe the resource in detail. The concept of URIs intends an application in various domains and therefore has only a few abstract rules. In general, URIs should be easy to communicate and understand. Therefore, human-readable terms and descriptions should be used. Analogue transmission and subsequent manual entry of URIs should also be taken into account. The use of characters, numbers and special characters is limited in order to ensure the most uniform and simple handling possible.
Depending on its purpose, URIs can be divided into Uniform Resource Location (URL)s and Uniform Resource Name (URN)s to apply to two different identification concepts:
Most addresses on the World Wide Web are referred to as Uniform Resource Location (URL) because they are describing a path to the server where the resource or service is located.
The term Uniform Resource Name (URN) is used to identify resources only by name regardless of their location. This ensures an unmistakable assignment of a name to a resource. Even if an assignment to one specific resource is not possible, the URN holds true for that abstract resource. URNs enable naming concepts, ideas, groups of resources,

etc. URNs are recognizable by their *urn* scheme.
URLs provide an answer *where* to find a resource, while URNs tell *who or what* a resource is.

2.2 Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is the fundamental protocol of the World Wide Web [20]. HTTP is a widely utilized standard for exchanging data as messages on the web. The protocol proposes how to format and transmit messages between various agents. This enables communication between client and server even though they do not know each other's exact implementation. They communicate via request from the client and corresponding response from the server. Each request from the client is independent of its predecessor or successor, and therefore the protocol is stateless. For a continuous flow of messages, additional technologies such as tokens or cookies have to be used. The protocol's semantics allow for a flexible content of the self-descriptive messages. HTTP uses URIs to identify resources like files or pages, their location often is expressed through URLs (Uniform Resource Location).
Every HTTP request contains a URI to define the target of the message and a method. The HTTP method defines the action to be performed on the target. GET (retrieve information), PUT (submit information), POST (update information), DELETE (delete information) are the most important methods for this work. Additionally, the request can contain a header. The header includes metadata that is needed to process the request or data that is submitted to the target for example.
A server has to respond to a client's request with at least one message. That response includes a status code to express the outcome of the response. For this work the important status codes are 200 (OK), 201 (created) and 401 (unauthorized). A response can have a header to communicate additional information to the client, such as files, metadata, etc.

2.3 Resource Description Framework (RDF)

The Resource Description Framework (RDF) is an abstract framework utilized to structure data on the web. The recent W3C Recommendation *RDF 1.1 Semantics* by *Hayes and Patel-Schneider* [23] is based on the W3C Recommendation *RDF Vocabulary Description Language 1.0: RDF Schema* by *Brickley and Guva* [11].
The Resource Description Framework (RDF) is a key concept in the field of knowledge representation and the semantic web. It forms the basis for the structured representation and linking of information on the World Wide Web. In this section, the essential aspects of RDF are explored, including its properties, syntax, semantics and practical applications. In addition, exemplary RDF datasets are presented to demonstrate the ability of

this framework to express and link knowledge. The core concept of RDF is to represent information in the form of triples consisting of subjects, predicates and objects. Each triple expresses a statement about a resource and its properties or relationships to other resources. RDF can be represented in a variety of syntaxes, including RDF/XML, Turtle, N-Triples and JSON-LD. In this thesis we will use Turtle. Each syntax allows RDF graphs to be expressed in different ways, while maintaining the basic structure of triples. The semantics of RDF are based on the idea that resources and their relationships are identified by unique URIs. URIs serve as globally unique identifiers for resources and allow information from different sources to be linked together. The use of URIs in RDF also allows the use of ontologies to define the meaning of resources and predicates. Ontologies are formal descriptions of knowledge domains that specify the semantics of concepts and relationships. RDF is widely used in various domains, including the semantic web, the Linked Data initiative, knowledge management and metadata modelling. It enables the integration and linking of data from different sources, leading to improved search engine performance, automation of data integration and knowledge discovery. RDF allows for modelling data that is machine-readable due to global definitions and graph structure, as well as being human-readable because of the triples reminding of sentences.

The example Listing 1 below describes the resource *:JohnDoe*. The subject of the triple is a local resource because it has no prefix. The triple in line 1 has the meaning "John Doe is a Person". The predicate *rdf:type* describes, that *:JohnDoe* is an instance of the RDF class *foaf:Person*. The prefix *foaf* is the short form for the URI of the Friend-of-a-Friend ontology, which defines predicates and classes in the domain of human relationships and human properties. Line 2 further describes *:JohnDoe* with a name that is represented by the literal *"John Doe"*.

Listing 1: RDF example

```
1 :JohnDoe   rdf:type    foaf:Person ;
2            foaf:name   "John Doe" .
```

2.4 Solid Project

The Solid Project is an initiative that was initiated by Sir Tim Berners-Lee, who invented the World Wide Web, and aims to strengthen the semantic web through privacy-oriented, decentralized technologies. This section presents the central concepts of the Solid Project and in particular the Solid Pods as the fundamental data structure. The Solid Project enhances users' sovereignty over their own data and opens up new possibilities for secure and collaborative data applications. The Solid Project was developed to address the problems of traditional data-centric web architectures. In traditional models, data is often tied

to specific services or applications, which can lead to a lack of data control and interoperability. The Solid Project aims to create an open and decentralized ecosystem where users have full control over their data and can share it securely with other applications and users.

Key objectives of the Solid Project are supporting data sovereignty and creating a semantic web. Solid enables users to store their data on a personal data server called Solid Pod, which is controlled by each user himself. This allows users to retain sovereignty over their own data and decide with whom they want to share it. By using Linked Data and RDF-based modelling, Solid Pods can be linked together and data can be structured logically and sensefully. This promotes interoperability and enables applications to better understand contextual information.

Solid Pods are the core of the Solid Project and serve as personal data containers for users. Each user can host their Solid Pod in an environment of their choice, whether on their own server or a third-party Solid server. The Solid Pod is identified by a unique URL. In a Solid Pod, users can structure and store their data in the form of Linked Data and RDF triples. This data can include personal information, contacts, calendar data, notes, files and much more. The use of RDF makes it possible to establish semantic relationships between data and thus achieve deeper meaning and interoperability. Solid Pods use security standards such as WebID for authentication of agents and ACL (Access Control Lists) to control the access to data. These mechanisms allow users to assign granular access rights and ensure that only authorized agents can access their data. [14]

2.5 Verifiable Credential Model v1.1

The Verifiable Credential Model v1.1 is a digital identity management technology for decentralized systems [29]. It was developed by the World Wide Web Consortium (W3C) and is a W3C recommendation aimed at changing the way digital identities and information about users can be verified and shared. In this thesis, the concept of the Verifiable Credential Model v1.1 is used to describe identities. It is used because this framework has the potential to improve the security, trustworthiness, and interoperability of digital identities on the semantic web and can support the change from centralized authentication servers to decentralized authentication technologies. The Verifiable Credential Model enables people to create trusted digital identities and use them in a decentralized system to prove their identity. Verifiable Credentials are issued by a third-party that is trusted by the verifier. By using digital signatures, the Verifiable Credential Model v1.1 enables a cryptographic verification of identity proofs. The issuing entity signs the credential with its private key and the verifier can verify the authenticity of the credential by checking the signature with the corresponding public key of the issuer. This reduces the risk of identity

fraud and data manipulation through faked identities. The framework also gives users full control over their own identity credentials. Users can choose what information they do and do not want to share. That supports data sovereignty of users on the semantic web.

An example of a Verifiable Credential (VC) is presented below in Listing 2 that represents information about an example student named "John Doe". The examples shown below only serve to illustrate the basic structure and elements of a VC and VP based on the Verifiable Credential Model v1.1 (VC Model). The content of the VC and VP may vary depending on the use case.

The VC consists of several parts [15]:

- **Subject:** The subject of the VC is the sample student itself, represented by the node name *<exampleStudent>*. Various attributes of the student are specified in the VC, such as the student's name, university membership, degree programme, degree sought, matriculation number and matriculation date. This information is later used to identify the student.
- **Description:** A description of the VC is given in the triple *schema:description* to briefly explain the content of the VC in more detail.
- **Holder:** The triple refers to an entity, typically an individual, that possesses and controls a Verifiable Credential. The holder of VC is the student, he can present and share the credential with a verifier or relying party to prove specific claims defined in the subject.
- **Validity period:** The VC has a validity period, which is specified by the triples *cred:validFrom* and *cred:expirationDate*. These dates indicate the period during which the VC can be considered authentic.
- **Issuer:** The issuer of the VC is specified in the triple *cred:issuer* and defined in the example as *ex:exampleUniversity*. The Verifier trusts the issuer and therefore considers the VC to be authentic.
- **Proof:** In the triple *sec:proof* a reference to the proof graph is given, which contains the information to verify the authenticity of the VC. In this example, the exact content of the proof graph is missing. Various technologies that enable a digital signature can be used for this purpose. For the implementation of the GDPR requirement to minimize data, the signing technology should support selective disclosure.

Listing 2: Verifiable Credential example

```
# file located at ex:exampleStudent/vc_1

<#exampleStudent> a schema:Person , ex:Student ;
    foaf:name                "John Doe" ;
    schema:memberOf          ex:exampleUniversity ;
    ex:degreeCourse          ex:MechanicalEngineering ;
    ex:degree                ex:BachelorOfScience ;
    ex:matriculationNumber   12345678 ;
    ex:enrolledSince         "2018-08"^^xsd:gYearMonth .

<#exampleStudentCredential> a cred:VerifiableCredential ;
    schema:description       "An example credential, issued by an example exversity
    to an example student." ;
    cred:credentialSubject   <#exampleStudent> ;
    cred:holder              <#exampleStudent> ;
    cred:validFrom           "2024-12-06T11:05:44.403Z"^^xsd:dateTime ;
    cred:expirationDate      "2023-12-06T11:05:44.403Z"^^xsd:dateTime ;
    cred:issuanceDate        "2022-12-06T11:05:44.403Z"^^xsd:dateTime ;
    cred:issuer              ex:exampleUniversity ;
    sec:proof                <#proofGraph> .

<#proofGraph> { ... }
```

The code example Listing 3 below shows a Verifiable Presentation (VP) for the student from the example Listing 2 above. A Verifiable Presentation is a special Verifiable Credential that is representing one or more attributes of a VC. The holder of the VP can thus prove that they are in possession of the VCs contained in the presentation and share the content of the VCs at the same time. The VP may also contain additional information or adjustments relevant to the specific use case. [29]

The VP consists of several parts [15]:

- **Verifiable Credential graph:** The VC attributes included in the presentation are listed in the *<#credentialGraph>* which is linked to the presentation in the triple *cred:verifiableCredential*. In the example, the subject of the VC shown earlier is referenced by the node *<exampleStudentCredential>*.
- **Holder:** The holder of the VP is specified in the triple *cred:holder* and is also defined as holder of the VC in the credential graph.
- **Proof:** Similar to the VC, the proof graph is referenced in the triple *sec:proof*, which

contains the information to verify the authenticity of the presentation. In addition, the proof graph of the contained VC is provided to check its authenticity as well, independent of the VP.

Listing 3: Verifiable Presentation example

```
# file located at ex:exampleStudent/vp_1

<#exampleStudentPresentation> a cred:VerifiablePresentation ;
    cred: verifiableCredential    <#credentialGraph> ;
    cred:holder                   <#exampleStudent> ;
    sec:proof                     <#presentationProofGraph> .


<#credentialGraph> {
    <#exampleStudentCredential> a cred:VerifiableCredential ;
        schema:description      "An example credential, issued by an example
    university to an example student." ;
        cred:credentialSubject  <#exampleStudent> ;
        cred:holder             <#exampleStudent> ;
        cred:validFrom          "2024-12-06T11:05:44.403Z"^^xsd:dateTime ;
        cred:expirationDate     "2023-12-06T11:05:44.403Z"^^xsd:dateTime ;
        cred:issuanceDate       "2022-12-06T11:05:44.403Z"^^xsd:dateTime ;
        cred:issuer             ex:exampleUniversity ;
        sec:proof               <#proofGraph> .

    <#exampleStudent> a schema:Person , ex:Student ;
        foaf:name               "John Doe" ;
        schema:memberOf         ex:exampleUniversity ;
        ex:degreeCourse         ex:MechanicalEngineering ;
        ex:degree               ex:BachelorOfScience ;
        ex:matriculationNumber  12345678 ;
        ex:enrolledSince        "2018-08"^^xsd:gYearMonth .
}

<#proofGraph> { ... }

<#presentationProofGraph> { ... }
```

In the following chapters, the examples of the access control system and the policies of the protocol will refer to the VP above.

For VPs, the selective disclosure functionality can be used to provide a flexible and privacy-

friendly way of sharing data, where the user retains control over their information and only shares data relevant to the specific situation. The functionality of selective disclosure is enabling the user to decide what information they want to disclose from their Verifiable Credentials without having to disclose the entire credential set. This increases the user's privacy and data sovereignty by giving them control over what information about them is and is not disclosed. Using selective disclosure in Verifiable Presentations allows the user to create targeted credentials that meet the verifier's specific needs. Let's imagine that a verifier only needs a person's matriculation status to grant access to certain services. Instead of sharing his entire Verifiable Credential set, the user can create a Verifiable Presentation containing only the matriculation attribute and submit it to the verifier. So the rest of his data remains private. [15]

2.6 Open Digital Rights Language (ODRL)

The Open Digital Rights Language (ODRL) is a standardized framework for modelling and managing rights of digital content. ODRL was developed to promote interoperability between different systems and platforms and to provide a common language for defining, communicating and implementing rights, conditions and constraints. The framework is independent of the platform or system on which the content is delivered. ODRL was also designed as a flexible framework that can support different application scenarios. It may also be extended by specific requirements and needs of individual applications. [24]

The ODRL language allows stating permissions, prohibitions and obligations over resources in a standardized RDF structure. Each ODRL profile policy consists of metadata to provide some context about the policy file itself. In addition, the policy contains at least one permission or prohibition that contains rules for a target resource [24].

The example in Listing 4 is a simple ODRL policy, it consists of one permission. The permission is a rule that allows an action on a target for a specific assignee. The policy could be extended with conditions or requirements that the assignee must fulfil.

Listing 4: ODRL example

```
ex:policy a odrl:Policy ;
    odrl:uid ex:policy ;
    odrl:permission [
        a odrl:Permission ;
        odrl:action          odrl:display ;
        odrl:target          ex:asset1 ;
        odrl:assignee        ex:user1 ;
        odrl:constraint      [ ... ] ;
        odrl:duty            [ ... ]
    ] .
```

Policies adhering to the ODRL profile can be of different types. An instance of *odrl:Offer* is meant for a wide audience that may not be known and proposes rules for the use of an target. An instance of *odrl:Agreement* represents rules that are granted between the required assigner and assignee parties. [24]

2.7 ODRL profile for Access Control (OAC) & DPV

The ODRL for Access Control (OAC) [17] provides an extension of the Open Digital Rights Language profile and is able to support the Access Control of a Solid Pod by extending the WAC specification (presented in section 4.1.1). OAC allows Pod owners to define complex access policies and rules for their resources, granting or revoking permissions to specific agents or groups. This is achieved by leveraging ODRL presented in 2.6, which enables the definition of policies using ODRL's vocabulary of actions, constraints, duties, and permissions. By extending ODRL to contribute to the access control system in Solid Pods, OAC ensures compatibility with existing ODRL ecosystems, ensuring interoperability and enabling the reuse of established policies.

One of the key advantages of OAC lies in its flexibility and adaptability. It supports a wide range of access control scenarios, from simple read and write permissions to more sophisticated conditional access rules. These access rules can be based on constraints such as time, location, or the relationship between agents and resources. Additionally, OAC allows for the modelling of default policies, which are used for access control decisions when specific policies are not explicitly defined. Additionally, OAC supports various ontologies to extend policies with additional information and more detailed or specific constraints. For this thesis espacially important is the extension of OAC with Data Privacy Vocabulary (DPV).

The Data Privacy Vocabulary (DPV) is a semantic ontology developed to model privacy concepts and information in RDF. The definitions of the ontologies terms are derived

from the GDPR definitions. This enables the consistent and standardized representation of privacy policies, consents, personal data and other privacy aspects. The DPV ontology enables one to integrate and process privacy information in applications and to apply privacy policies that are unambiguous and transparent. [26]

3 Related Work

The analysis of related work in this chapter serves to identify the current state of knowledge in the research area of semantic web and decentralized access control systems. By examining previous studies and findings, we can describe the existing state of knowledge, identify research gaps and gain a better overview of solutions for decentralized access control mechanisms. In addition, our approach can be better contextualized in this thesis and advances on existing technologies can be made clearer.

We respect the work of the researchers who wrote the following papers and acknowledge that this thesis is based on the efforts of the following authors, in the sense that ideas, concepts and models are inspired by the related papers.

First, *Usage Control Policy (UCP)* by Akaichi and Sirrane in section 3.1 is presented. After, the *DIF Presentation Exchange* specification in section 3.2 is presented. Then, the work of Braun and Käfer *Attribute-based Access Control on Solid Pods using Privacy-friendly Credentials* is presented in section 3.3. Finally, *SISSI: An Architecture for Semantic Interoperable Self-Sovereign Identity-based Access Control on the Web* by Braun, Papanchev and Käfer is presented in section 3.4.

3.1 Usage Control Policy (UCP)

The paper of Akaichi and Kirrane introduces a new policy language called Usage Control Policy (UCP) for managing data usage in decentralized systems like IoT and distributed knowledge graph applications [2]. It addresses the need for policy languages that can express normative statements, permissions, prohibitions, and obligations related to data use with fine-grained conditions. The proposed UCP language builds upon existing policy languages like ODRL and Rei but extends their capabilities to support more specific usage control requirements.

The use case presented in the paper involves a smart city scenario where marketing companies want to access data from smart objects, and the manufacturers of these objects want to enforce usage policies on how the data can be used. The UCP language allows policies to be expressed using duties, obligations, constraints, actions, and nested rules to cover various policy structures.

The paper demonstrates the encoding of two usage control policies using the UCP language, one involving the permission for downloading power consumption data with an obligation to store it for a limited time, and the other allowing data download for aggregation purposes only.

Future work includes exploring more fine-grained conditions and monitoring the fulfillment

of obligations by end-users using states of deontic concepts. Additionally, the authors plan to study how logic structures of policies can be modelled to be simple and easily readable.

3.2 DIF Presentation Exchange

The DIF Presentation Exchange specification aims to standardize the exchange of identity information in decentralized identity systems [12]. The Presentation Exchange specification only defines a data model, implementation is left open by using any JSON format of choice. The specification provides a technology to have identity claims without being reliable of centralized servers or third parties. The only time a third party is needed, is when issuing the claims. Verifiable claims can include attributes, credentials, or proofs about individuals or entities. Additionally, the specification presents a mechanism to present the claims from a Holder to a Verifier. Finally, DIF Presentation Exchange enables users to define requirements on the presented claims.

The authors present a data model for the "Presentation Definition". The Verifier can define which attributes or claims he requires the Holder to present. With standardized generic descriptions of claims the Verifier is able to communicate his requirements in a simple to read and transparent manner by sharing the Presentation Definition object with the Holder. The Holder can decide whether he proceeds and presents his claims or not. In order to standardize the data model of a Presentation Definition, the specification defines which properties the presentation definition must contain and which data types must be used for it. The specification defines numerous extensions to the constraints for claim attributes, and thus enables fine-grained conditions to be imposed on presentations.

"Presentation Submissions" are objects that describe the form in which the claims presented by the Holder are delivered to the Verifier. The submission is based on the presentation definition and thus facilitates the interpretation of the data for the verifier. Again, the specification determines which properties the presentation submission must contain and which data types must be used for the properties.

Moreover, the purpose of a submission is also to enable the validation of claims. The data embedded in the submission is matched against the presentation definition to validate the content of the presentation.

3.3 Attribute-based Access Control on Solid Pods using Privacy-friendly Credentials

Braun and Käfer present a proof-of-concept demo that provides users with the ability to request Verifiable Credentials from other users, store the credentials on a solid pod and use credentials for authentication and authorization purposes on other solid pods [9]. The demo is based on servers adhering to the Solid Protocol[14] and the Verifiable Credentials conform with the Verifiable Credential Model[29].

The demo is presented in the author's work by a walkthrough. The accessing user identifies himself using their WebID, as it is done on Solid Pods[14]. First, the user tires to access the data of the data provider and gets rejected by the server. A Linked Data Notification is sent from the server to the accessing user, containing the requested attributes of the credential. The accessing user then creates a Verifiable Credential that proves the attributes. In order to gain access, the user needs to send a Verifiable Presentation of his credentials to the inbox of the solid pod of the data provider. He doesn't need to fully disclose his credential, as selective disclosure mechanisms in the demo allow for only sharing required attributes. That ensures data minimization. A server module then validates the Verifiable Presentations regarding issuer, necessary attributes and validity of the signature. If the Presentation meets the server's requirements, the user is granted access via his WebID.

The secure handling of personal data is not addressed in this work. And another weakness is, that the user needs to disclose his WebID.

This thesis is influenced by the paper, as it originates from the same institute and concepts are therefore taken up in this work.

3.4 SISSI: An Architecture for Semantic Interoperable Self-Sovereign Identity-based Access Control on the Web

In their work, the authors Braun, Papanchev and Käfer present a framework that enables authentication and authorization based on self-sovereign identity [10]. They use the Verifiable Credential Model based on RDF and define Access Control Rules with SHACL. The use of widespread standards should ensure interoperability between semantic technologies. In the paper, the usefulness of the presented protocol is explained by means of a use case. A student is supposed to access data of his university and has to present a verifiable credential that meets the conditions of the university. Since the VC is decentralized at the student's site and digitally signed by the university, authentication and authorization can be carried out directly on the web server without a central server. To check the conditions, the web access control is extended with a SHACL shape, which is

then matched against the Verifiable Presentation.

The paper presents the architecture of a system that implements the presented protocol and can realize the presented use case. This architecture is evaluated by means of a comparison with blockchain-based solutions. The authors' web-based solutions have a lower execution time than blockchain solutions.

The research gap we identified is the lack of securing confidential handling of personal data and a transparent communication of the purpose of the processing, adhering to the GDPR, is not adressed in this paper.

This thesis is influenced by the paper, as it originates from the same institute and concepts are therefore taken up in this work.

4 Data Models

4.1 Access Control Languages

This section will provide an overview over access control languages used to model access control systems in decentralized linked data systems.

The term 'Access Control System' describes a security system of a data storage system. In a decentralized linked data system like Solid, an access control system is used to define and enforce access rules for resources. An access control system ensures integrity and availability of data for certain agents or agent groups.

In a system without any access control, resources are likely to be accessed, changed, deleted or created by unauthorized users. That could lead to data breaches, data loss or unwanted modification. [13][28]

Access control rules adhere to a certain data model. The data model for Access Control Lists (ACLs) provides a generalized standard for more fine-grained Access Control Rules. ACLs are lists which describe rules for the access to resources. There can be a list for each resource in a data storage system. ACLs can independently be applied to every level (e.g. files, directories, networks, systems) of a system. A sublevel can inherit Access Rules from its parent level. Every item of an ACL contains at least one allowed action and related authorized agents. An agent must be a person, social entity or software identified by a URI. Agents can be grouped or described by a class. [13]

4.1.1 Web Access Control (WAC)

Web Access Control (WAC) is a W3C Candidate Recommendation. It is a framework for decentralized linked data platforms to implement an access control system for HTTP resources. WAC utilizes the ACL model to express access control rules with RDF. WAC is the framework used in the Solid Project. If a user requests a web resource via the HTTP framework, the Solid Pod responds with a link header to the corresponding ACL resource as link target. As a result, the client now can discover the ACL resource by checking the *rel* parameter of the link header. The ACL resource is an RDF document and contains the access rules for the requested web resource. The access rules are modelled with RDF statements using the ACL ontology. [13]

Listing 5: ACL example from a Solid Pod

```
# ACL resource for the public folder
@prefix :      <#> .
@prefix pub:   <./> .
@prefix c:     </profile/card#> .

# Everyone can append and read
:AppendRead
    a acl:Authorization ;
    acl:accessTo     pub: ;
    acl:agentClass   foaf:Agent ;
    acl:default      pub: ;
    acl:mode         acl:Append, acl:Read .

# The owner has all permissions
:ControlReadWrite
    a acl:Authorization ;
    acl:accessTo     pub: ;
    acl:agent        c:me ;
    acl:default      pub: ;
    acl:mode         acl:Control, acl:Read, acl:Write .
```

The example above represents a typical access control list of a Solid Pod. The set of rules regulates the access to the public directory. Every rule is modelled as a local blank node and recognizable as an access rule by being an instance of class *acl:Authorization*.
The upper rule *:AppendRead* states that everyone that is an entity of class *foaf:Agent* is able to create (*acl:Append*) and read (*acl:Read*) resources in the public directory. In other words: "everyone" can read the public directory.
The lower rule *:ControlReadWrite* regulates the access to the same public directory. The owner (*c:me*) can move, delete (*acl:Control*), read (*acl:Read*) and create, edit (*acl:Write*) resources in the public directory. The owner respectively has all access rights to manage his resources and needs to authenticate with the *c:me* URI. [13]

The example presents that it is only possible to authenticate clients through a URI or an agent class. If more detailed conditions are to be defined on client properties, the functions of WAC are not sufficient. WAC is therefore not suitable for verifying an identity by checking VC attributes, because the conditions can not be modelled.

The consequences of this lack of functionality are explained in section 4.3.1.

4.1.2 Access Control Policy Language (ACP)

The Access Control Policy Language (ACP) Specification is an Editor's Draft by the W3C Solid Community Group [8]. ACP is a language to express rules for resource usage on a Solid Pod. ACP is expected to be the successor or supplement the current access control system of solid pods explained in the previous section 4.1.1.

An ACP resource consist of several components which are linked to each other: The context graph represents metadata about the access request of a client. For example the target resource, the requesting agent, the owner of the data, etc. can be modelled with a context graph. Each access of a client is represented by an instance of the class *acp:Context*.
Authorization graphs represent the access rule for a resource. An instance of the class Access Control Resource links the resource to more detailed access control graphs that will be applied to the resource if a client requests access.
Instances of the Access Control class represent an overview of the Policies that are linked to the Access Control Resource and are applied to them.
Policies link Access Controls to Matchers and apply an Access Mode to certain Matchers. If an accessing user satisfies the matcher the policy allows or denies acl Access Modes (see 4.1.1).
Matchers are resources that represent conditions that accessing users have to fulfil. Conditions can be URIs or other RDF attributes. Additionally, VCs can defined as constraints to fulfil.

The specification [8] mentions that Matchers can be satisfied when an accessing user presents a valid VC that is type of the defined credential. However, there is no further information on how attributes are matched against the credential and to what extent detailed attributes of credentials are checked. When we asked the ACP community how these functions will be implemented and developed in the future, we did not receive an answer. For this thesis, we must therefore assume that VC handling is not sufficiently implementable for our use case.

The consequences of this assumption are explained in section 4.3.1.

4.2 Constraint Languages

In this chapter, we delve into the topic of constraint languages for RDF graphs. Constraint languages are used to define graphs, which are referred to as 'shapes', that describe another graph. A shape defines rules for the structure and content of an RDF data graph and is used to enforce the rules when validating the RDF graph against the constraints. The main constraint languages used for RDF data are SHACL and ShEx, which will be presented in the two following sections. They both provide similar mechanisms to Each constraint language has its own strengths and weaknesses and therefore the selection is highly dependent on the use case. The goal of this chapter is to make a choice between SHACL and ShEx to model required VC attributes for an access rule. [3][25][21]

4.2.1 SHACL

The Shape Constraint Language (SHACL) is a language and framework in the field of semantic web technologies [25]. SHACL is used to define structures and validate rules for RDF data. SHACL offers the possibility to not only describe content of a RDF graph, it can check the content and completeness of RDF data. SHACL was developed as part of the W3C (World Wide Web Consortium) Data Shapes Working Group and was published as a W3C Recommendation in March 2017. It builds on the SPARQL standard and allows the formal description of conditions that RDF graphs must fulfil. These conditions are called "shapes" and specify which triples should be present in an RDF graph and what values these triples may have. A typical application example is checking correctness of data in an RDF graph with a shape. Here SHACL can be used to ensure that certain properties are present for certain classes, that data types are correct, or that relationships between URIs have been modelled correctly. SHACL also provides advanced functions for checking cardinalities, count and more.

A SHACL shape consists of two main components: a node shape and a property shape. The node shape defines the constraints for the nodes in the RDF graph, while the property shape further defines the constraints for the predicates and objects of the nodes.

Below in Listing 6 is an example of a simple SHACL shape. In the following the composition of node shape and property shape is explained. The example shape first defines for which RDF subjects to search for. Therefore the triple *ex:exampleShape sh:targetClass cred:VerifiableCredential* defines that the subject needs to be an instance of class *cred:VerifiableCredential*. The properties of the desired node are defined with *sh:property* in the blank node. The predicate of the node should be *cred:credentialSubject* defined by the *sh:path* term. The object of the desired node should be any kind of an IRI.

Listing 6: SHACL shape example

```
ex:exampleShape a sh:NodeShape ;
    sh:targetClass cred: VerifiableCredential  ;
    sh:property [
        sh:path cred:credentialSubject  ;
        sh:nodeKind sh:IRI
    ] .
```

The example data graph in Listing 7 satisfies the shape above because it contains a node that has the required properties. A validation of the data graph against the shape would produce a validation report as seen in Listing 8, which describes the validation as successful by *sh:conforms* being *true*. The validation result provides information about which node was matched with which shape.

Listing 7: Data graph example

```
ex:credentialGraph a cred: VerifiableCredential  ;
    cred:credentialSubject ex:Student .
```

Listing 8: SHACL validation result example

```
[
    a sh:ValidationReport ;
    sh:conforms          true  ;
    sh: result           [
        a sh:ValidationResult ;
        sh:focusNode       ex:credentialGraph ;
        sh:sourceShape     ex:exampleShape ;
    ]
] .
```

4.2.2 ShEx

Shape Expressions (ShEx) is a formal language and semantic framework developed by a Community Group to enable forming schemas and validation of RDF graphs [27]. ShEx provides a declarative way to describe complex patterns in RDF graphs and to check whether these patterns are satisfied by given RDF instances. ShEx makes it possible to define patterns that must be fulfilled by RDF graphs. As with SHACL, these patterns are also referred to as "shapes" and can describe both simple properties and complex relationships between resources. ShEx provides a compact, human- and machine-readable, and precise way to describe the structure of RDF graphs. ShEx uses an abstract ASCII-

based syntax which is inspired by Turtle. The syntax includes constructs such as node labels, cardinalities, and logical operators to express complex conditions on RDF graphs. Like SHACL, ShEx provides mechanisms for validating RDF graphs against a given ShEx schema. The validation is done in a process of successively matching the RDF graph against each shape. A ShEx shape consists of a schema, which models the conditions on an RDF graph, and a shape map, which describes which nodes are to be examined with the shape map. A shape map can be a fixed shape map if it describes a certain node or can be a query shape map when the node is unknown and has to be searched in the RDF graph.

The example Listing 9 represents the same requirements as the SHACL shape in the form of a ShEx schema. The schema describes the desired properties of the node. The query shape map defines which node to search for in the RDF graph. The node needs to be an instance of *cred:VerifiabelCredential*. The node would fill the subject position *FOCUS*. To that node the exampleShape should be validated against.

Listing 9: ShEX shape example

```
### schema
start = @<exampleShape>
<exampleShape> {
    cred:credentialSubject      IRI .
}

### query shape map
{FOCUS rdf:type cred:VerifiableCredential}@<exampleShape>
```

4.2.3 Comparing SHACL to ShEx

- **Purpose:** Although both SHACL and ShEx have the same functionality in structuring and verifying RDF data, the approaches are quite different. ShEx focuses on describing RDF graph structures like a grammar or schema, allowing validation against those descriptions. It provides a way to represent the structure of RDF graphs. ShEx is rather used to have a blueprint for RDF graphs that are not yet existing. Certain data formats utilizing RDF may express their structure or schema using a ShEx shape.
  SHACL is rather made for verifying existing graphs against a shape. SHACL is designed to be a constraint language that enables a detailed description of modelling a set of constraints for RDF graphs, for which the more compact syntax is helpful. As the constraint shape in this thesis is used for validating the presented VP, SHACl fits our requirements better. [21]

- **Interoperability:** SHACL is a W3C Recommendation and therefore is a more widely accepted standard to describe and validate RDF data [25]. That means in context of interoperability, SHACL is more evolved and there is community support available for SHACL. ShEx is a community effort under the W3C Community Contributor Licence Agreement by the Shape Expression Community Group [27]. It is not that widely adopted, and the number of compatible tools and libraries is only just arising [21].

- **Syntax:** The syntax of SHACL are regular RDF triples to defines shapes and can therefore be expressed in turtle [25]. This enables a uniform semantic for all components of the protocol. Additionally, SHACL uses the RDF abstract syntax, more complex shapes can be formed and there is more information available about well-formed SHACL shapes [21]. ShEx uses a compact syntax based on turtle, which allows for a short modelling of a shape and can contribute to easier readability by a user [27].

- **Validation result:** The quality of the validation results generated also varies. SHACL can generate comprehensive validation reports that provide detailed information about all errors and warnings. It can also generate "validation reports" that allow the results to be processed in machine-readable form [25]. ShEx also provides a clear error message if the data does not match the defined shape expressions. The results are usually easy to interpret and allow for quick troubleshooting. For successful matching, however, ShEx does not provide a detailed report, but only the information that the data graph conforms to the shape [3].

4.3 Access Rule data model

The initial step of modelling a Verifiable Consent involves modelling access control rules. Through this modelling, we aim to create a structured approach that clearly outlines who can access the data and under what conditions, ensuring that the data providers have control over their resources.

This section covers the modelling of *Access Control Rules*. First, the basic access control requirements for the *Verifiable Consent Protocol* are presented. We present examples of the possible implementation of authentication using the Verifiable Credential Model (2.5). In section 4.3.1, the implementation of access rules for Linked Data Platforms follows. For this purpose, a modification is made to the access rules of a Solid Pod on the basis of WAC and thus an *Access Rule Shape* is integrated. This *Access Rule Shape* is presented in section 4.3.2. SHACL is used for the shape, but an alternative implementation with ShEx is also presented. The advantages and disadvantages of each language have already been discussed in section 4.2.3.

Requirements for an access control system of a Linked Data platform that wants to use Verifiable Credential (VC) for authentication could be summarized as follows:

1. **Interoperability:** Technologies used to implement the access control system should be standardized and widely accepted. This allows the protocol to be implemented for many different use cases, as a high level of standardization allows for a high level of interoperability. For this purpose, the technologies of the semantic web are already well suited, as they offer a simple interpretation of data for humans and machines through a self-description and can process other standards and protocols of the semantic web. [5]

2. **Easy integration into existing systems:** We suggest, the access control system should allow easy integration into existing access control systems without having to make extensive adjustments. This will facilitate the adoption of the Verifiable Credentials technology, as organizations can maintain their existing infrastructures and processes while gradually benefitting from the advantages of Verifiable Credentials.

3. **Granular access control:** The system should enable granular access control to set fine-grained permissions for different resources and actions. Therefore, we suggest utilizing the Access Control List (ACL) model.

4. **Decentralization:** We suggest, the access control system should support the decentralized character of Linked Data. It should be able to accept and process Verifiable Credentials from different identity providers to enable interoperable authentication.

4.3.1 Access Control utilizing WAC & ACL

Taking into account the characteristics of the technologies presented in the previous chapters, we conclude that none of WAC, ACP, SHACL or ShEx individually are able to model access control rules based on VCs. But combining the desired features of the policy languages and constraint languages can lead to an intended result. We rely on the structure of ACLs for defining the access control rule on the Solid Pod and therefore utilize the Access Control List Ontology. Instead of defining who is permitted to access by using *acl:agent* or *acl:agentClass*, we enable the data provider to model VC attribute based access control rules by extending the WAC with ACP elements and a SHACL shape. To express that the client needs to present the required VP attributes, we use the *acp:vc* predicate within the access rule. The corresponding object is a SHACL shape that describes a Verifiable Presentation.
The server must grant access to the resource only when the presented VP matches the SHACL shape specified in the ACR. The client is then able to read the information provided in the target resource.

The ACL rule in the example below is defined for the "public" folder, as indicated by the comment. The example consists of the following characteristics:

The example begins with the definition of namespace prefixes using the *@prefix* statements. These prefixes allow for a more concise representation of URIs. In this case, the prefixes used are as follows:

- *:* represents the base URI of the current document.
- *pub:* represents the relative URI *./* which refers to the current directory.
- *c:* represents the URI */profile/card*.

Please note that all ontology prefixes are omitted in this example and can be found in a table (not included here).

The first authorization rule, labelled as *:Read*, grants permission to read, therefore the access mode for this rule is set to *acl:Read*. It is of type *acl:Authorization* and applies to the resource identified by *pub:* (the public folder). This rule serves as the default rule *acl:default* for new child resources of the public folder. To access the resource, it requires a Verifiable Presentation (VP) that matches the shape specified by the resource *ex:exampleShape#VerifiablePresentationShape* (Listing 11). The access rule references data usage policies that are applied to the resource

Listing 10: ACL example for an Access Control Rule

```
# ACL resource for the public folder
@prefix :      <#> .
@prefix pub:   <./> .
@prefix c:     </profile/card#> .

# Permission to read when presenting a VP matching
    ex:exampleShape#VerifiablePresentationShape
:Read
    a acl:Authorization ;
    acl:accessTo     pub: ;
    acl:default      pub: ;
    acp:vc           ex:exampleShape#VerifiablePresentationShape ;
    odrl:hasPolicy   ex:offerR, ex:requestVP ;
    acl:mode         acl:Read .
```

Our approach of validating a VC in an access rule independently of webIDs is not yet to be found in any of the access control language specifications. There is an attempt to use VCs with ACP, but the function is only briefly mentioned in [8] and is not explained in further detail. When asked in the ACP community, no further information on the future development of this feature was shared. We therefore assume in this thesis that ACP does define a predicate to integrate VCs into access rules. However, we assume the function of validation is not possible with ACP. Should a validation of VC attributes be possible in future publications, WAC could be replaced by ACP and possibly even the use of constraint languages could be dispensed with.

It is important to mention that Web Access Control needs to be modified to enable a matching of SHACL shapes in general and features have to be added to the Solid Pod to procedure the protocol presented in the following. The necessary changes to the access control system and Solid Pod are beyond the scope of this thesis.

4.3.2 SHACL shape

In this section, the SHACL shape mentioned in Listing 10 is described. As the ACL ontology is not able to express explicit constraints based on VC attributes for agents, a constraint language is needed to do that. In this thesis, we will focus on SHACL for the reasons given in section 4.2. However, ShEx could also be used to model credential requirements, so in the course of this section an example of the alternative implementation with ShEx will be briefly presented.

The SHACL shape is modelled by the data provider. He defines which VC attributes are needed to be presented in order to access his data. In an academic context, that may be the proof of matriculation, as seen in the example below. The example Listing 11 below demands the fulfilment of a shape attribute *[] a ex:Student .* The Verifiable Presentation (VP) Listing 3 in section **??** satisfies this requirement of being a student, because of the matching triple *<#exampleStudent> a ex:Student .* in line 3.

The shape definition labeled as *VerifiablePresentationShape* defines the constraints for instances of the *cred:VerifiablePresentation* class. It ensures that a Verifiable Presentation must have at least one *cred:verifiableCredential* property, pointing to the Verifiable Credential Graph. Additionally, it mandates the presence of a single *cred:holder* property (referring to the entity holding the VP) and at least one *sec:proof* property (representing the cryptographic proof of the VP's integrity).

The Verifiable Credential Graph is defined by the shape labeled as *VerifiableCredentialGraphShape*. It specifies constraints for instances of the *cred:VerifiableCredentialGraph* class. According to the shape, a Verifiable Credential Graph must have at least one *cred:credentialSubject* property, pointing to an entity represented by an IRI. Within this property, there is a nested shape constraint that requires the presence of an *a* property (representing the RDF type) with a fixed value of *ex:Student*.

Furthermore, the Verifiable Credential Graph must have a single *cred:holder* property, indicating the entity holding the credential. It should also include a single *cred:issuanceDate* property with a value of type *xsd:dateTime* (representing the date and time of credential issuance). The *cred:validFrom* property must have a value less than the *cred:expirationDate* property. Speaking of which, the *cred:expirationDate* property must have a value of *xsd:dateTime* data type.

Additionally, the Verifiable Credential Graph must have at least one *cred:issuer* property, pointing to the entity issuing the credential. Lastly, it should have at least one *sec:proof* property to ensure the inclusion of cryptographic proof.

The SHACL shape represents a standardized Verifiable Presentation introduced in section 2.5. To change the required VC attribute, it is only needed to change lines 30 & 31

regarding the property of the credential subject and the line 61 regarding the issuer of the credential. All other elements of the shape simply define that an RDF graph is needed to be presented that satisfies the structure of a valid VP and that this graph contains all the properties defined by the Verifiable Credential Model v1.1 (VC Model). [29]

Listing 11: SHACL example used in Access Control Rule

```
# file located at ex:exampleShape
<#VerifiablePresentationShape>
    a sh:NodeShape ;
    sh:targetClass  cred: VerifiablePresentation  ;
    sh:property     [
        sh:path         cred: verifiableCredential  ;
        sh:node         <#VerifiableCredentialGraphShape> ;
        sh:minCount     1
    ] ;
    sh:property     [
        sh:path         cred:holder ;
        sh:nodeKind     sh:IRI ;
        sh:minCount     1 ;
        sh:maxCount     1
    ] ;
    sh:property     [
        sh:path         sec:proof ;
        sh:minCount     1
    ] .

<#VerifiableCredentialGraphShape>
    a sh:NodeShape ;
    sh:targetClass  cred:VerifiableCredentialGraph ;
    sh:property     [
        sh:path         cred:credentialSubject  ;
        sh:nodeKind     sh:IRI ;
        sh:minCount     1 ;
        sh:node         [
            sh:property     [
                sh:path         a  ;
                sh:value        ex:Student
            ]
        ]
    ] ;
    sh:property     [
```

```
36        sh:path          cred:holder ;
37        sh:nodeKind      sh:IRI ;
38        sh:minCount      1 ;
39        sh:maxCount      1
40    ] ;
41    sh:property      [
42        sh:path          cred:issuanceDate ;
43        sh:datatype      xsd:dateTime ;
44        sh:minCount      1 ;
45        sh:maxCount      1
46    ] ;
47    sh:property      [
48        sh:path          cred:validFrom ;
49        sh:lessThan      cred:expirationDate ;
50        sh:minCount      1 ;
51        sh:maxCount      1
52    ] ;
53    sh:property      [
54        sh:path          cred:expirationDate ;
55        sh:datatype      xsd:dateTime ;
56        sh:minCount      1 ;
57        sh:maxCount      1
58    ] ;
59    sh:property      [
60        sh:path          cred:issuer ;
61        sh:node          ex:exampleUniversity ;
62        sh:minCount      1
63    ] ;
64    sh:property      [
65        sh:path          sec:proof ;
66        sh:minCount      1
67    ] .
```

The ShEx example 12 below describes the same RDF graph structure as the SHACL shape 11. It would also be successfully matched with <exampleStudent> in VP 3. The discussion whether to use SHACL or ShEx is found in section 4.2.3.

Listing 12: ShEx example

```
start = @<PresentationShape>

<PresentationShape> {
    a [ cred: VerifiablePresentation ] ;
    cred: verifiableCredential    @<CredentialShape> ;
    cred:holder                   IRI ;
    sec:proof                     IRI
}

<CredentialShape> {
    a [ cred:VerifiableCredentialGraph ]
    cred:credentialSubject   @<CredentialSubjectShape> ;
    cred:holder              IRI ;
    cred:expirationDate      xsd:dateTime ;
    cred:issued              xsd:dateTime ;
    cred: issuer             ex:exampleUniversity ;
    cred:validFrom           xsd:dateTime ;
    sec:proof                IRI ;
}

<CredentialSubjectShape> {
    IRI a ex:Student .
}
```

4.4 Non-Personal Data Usage Policy data model

We model non-personal data usage policies for common non-abstract resources like files or directories according to the Open Digital Rights Language (ODRL) profile [24]. We aim to follow standardized approaches where possible to ensure a wide range of interoperability. Additionally, in the context of data processing, transparency and simple language are particularly important to facilitate user understanding. Using standards which are already used in practice and whose properties are defined precisely and in detail, enables the user an easier and quicker understanding to define and interpret policies. Moreover, the regulation of the GDPR states an important guiding principle in *Article 5 1.) a GDPR* and demands the following requirement for the processing of personal data:
"Personal data shall be processed lawfully, fairly and in a transparent manner in relation to the data subject. [...]" [18]
Although the resource, the handling of which is defined with the following policy, not necessarily is personal data, we have the ambition to limit or expand the processing of the resource as precisely as possible.

Within the Verifiable Consent Protocol 5.1 (see overview Figure 1), the *Policy R* is a data usage policy. It defines the rules for the usage of *Resource R*. If the client tries to access the target resource, the server responds with the *Policy R Offer* in the HTTP header (see Figure 2). That *Policy R Offer* is an *odrl:Offer* and therefore follows the ODRL definition (see line 2 in Listing 14). An ODRL offer is used when stating rules for a resource that is available for a large number of recipients whose individual identity may not be known [24]. That fits the idea of a decentralized VC attribute based access control system very well, since the client is only known in a way, that he is a member of a group to which a specific credential has been issued. Explained using the example in this thesis: The university only wants to verify if the client is a student. The university does not know which of the students accesses the provided data, neither they are allowed to ask for information that is irrelevant for the purpose of proving their matriculation status. So the only VC attribute they are asking for, is the triple *[] a ex:Student* defined in the SHACL shape (see Listing 11).

The ODRL profile policy *ex:OfferR* in the provided example 14 below focuses on granting permission to read teaching content for the purpose of academic research. Since the server sends the offer to the client, the creator *ex:exampleUniversity* is mentioned in the metadata at the top of Listing 14.

The permission section of the policy states that *ex:exampleUniversity*, as the *odrl:assigner*, grants the action of reading (*odrl:Read*) to anyone who satisfies the constraints and has the target of *ex:exampleTeachingContent* (*Resource R*). The enbedded access control rule specifies that the permission is granted specifically for reading (*acl:Read*).

The policy includes a constraint that further refines the permission. The constraint specifies that the purpose (*dpv:Purpose*) of the granted permission should be academic research (*dpv:AcademicResearch*). This ensures that the granted access is intended specifically for academic research purposes related to the teaching content.

In addition to the permission, the policy includes a duty. The duty states that the action of granting use (*odrl:grantUse*) is required. The duty's target is *_:VerifiablePresentation*, which represents a verifiable presentation. The duty also includes a constraint that specifies that the grantee of the use should be *ex:exampleUniversity*. This constraint ensures that only *ex:exampleUniversity* can exercise the granted use and process the Verifiable Presentation.

Furthermore, the *_:VerifiablePresentation* requested in the duty is described in more detail. With the help of a blank node, a placeholder for the later VP is created here. The VP does not yet exist at the time of the creation of Policy A Offer and even if it did, it would not yet be known to the server. The server cannot therefore use an explicit IRI to reference the VP. Instead, the blank node is used. This will be described in more detail:
Lines 36-44 represent a result of a validation of an RDF graph using SHACL. The validation report states that the blank node *_:VerifiablePresentation* must conform to the shape *ex:exampleShape#VerifiablePresentationShape*. That ensures, the client knows he needs to provide not any VP, but the one which satisfies the shape and provides information about his identity.
Line 46 defines that the blank node is described in more detail by Policy VP Request. The Policy VP Request is sent to the client at the same time as the Policy R Offer.

Listing 13: Policy R Offer - offers rules for target resoure

```
1  ex:offerR
2      a odrl:Offer;
3      odrl: profile          odrl: ;
4      dc:description         "Offer to read Teaching Content for Academic Research
       purposes." ;
5      dc:creator             ex:exampleUniversity ;
6      dc:issued              "2022-11-08T17:58:31"^^xsd:dateTime ;
7      dc:valid               "P1Y0M0D0H0S"^^xsd:duration ;
8      odrl:uid               ex:offerR  ;
9      sec:proof              <#proofGraph> ;
10     odrl:permission        [
11         odrl:assigner          ex:exampleUniversity ;
12         odrl:action            odrl:Read ;
13         odrl:target            ex:exampleTeachingContent ;
14         acp:grant              acl:Read ;
```

```
15        acp:context          [
16            acp:target           ex:exampleTeachingContent
17        ] ;
18        odrl:constraint      [
19            a odrl:Constraint ;
20            dc: title            "Permission to read teaching content for the
    purpose of academic research." ;
21            odrl:leftOperand     dpv:Purpose ;
22            odrl:operator        odrl:isA ;
23            odrl:rightOperand    dpv:AcademicResearch
24        ] ;
25        odrl:duty            [
26            a odrl:Duty ;
27            odrl:action          odrl:grantUse ;
28            odrl:target          _:VerifiablePresentation ;
29            odrl:constraint      [
30                a odrl:Constraint ;
31                dc: title            "Grantee of use is exampleUniversity." ;
32                schema:grantee       ex:exampleUniversity
33            ]
34        ]
35    ] .
36
37  [
38      a sh:ValidationReport ;
39      sh:conforms          true ;
40      sh:result            [
41          a sh:ValidationResult ;
42          sh:focusNode         _:VerifiablePresentation ;
43          sh:sourceShape       ex:exampleShape#VerifiablePresentationShape ;
44      ]
45  ] .
46
47  _:VerifiablePresentation odrl:hasPolicy ex:requestVP .
48
49  <#proofGraph> { ... }
```

After the client receives the *Policy R Offer*, he creates the *Policy R Agreement* if he accepts the terms. The content of the *Policy R Agreement* described in example 14 below is therefore almost the same. The creator changes, as the student creates the agreement. And he adds himself in line 13 as assignee of the policy and in line 18 as accessing agent.

So it is now clear for whom the policy is explicitly intended and to whom read access is granted. Previously, no explicit IRI could be added to the offer for either of these, as the identity of the client was not yet known to the server.

Listing 14: Policy R Agreement - agrees to rules for target resoure

```
ex:agreementR
    a odrl:Rgreement;
    odrl: profile          odrl: ;
    dc:description "Agreement between an University and a Student to read Teaching
    Content for Academic Research purposes." ;
    dc:creator             ex:exampleStudent ;
    dc:issued              "2022-11-08T17:58:31"^^xsd:dateTime ;
    dc:valid               "P1Y0M0D0H0S"^^xsd:duration ;
    dc:references          ex:offerR  ;
    odrl:uid               ex:agreementR ;
    sec:proof              <#proofGraph> ;
    odrl:permission        [
        a odrl:Permission ;
        odrl:assigner          ex:exampleUniversity ;
        odrl:assignee          ex:exampleStudent ;
        odrl:action            odrl:Read ;
        odrl:target            ex:exampleTeachingContent ;
        acp:grant              acl:Read ;
        acp:context            [
            acp:agent              ex:exampleStudent ;
            acp:target             ex:exampleTeachingContent
        ] ;
        odrl:constraint        [
            a odrl:Constraint ;
            dc: title "Permission to read teaching content for the purpose of
    academic research." ;
            odrl:leftOperand     dpv:Purpose ;
            odrl:operator        odrl:isA ;
            odrl:rightOperand    dpv:AcademicResearch
        ] ;
        odrl:duty              [
            a odrl:Duty ;
            odrl:action            odrl:grantUse ;
            odrl:target            _:VerifiablePresentation  ;
            odrl:constraint        [
                a odrl:Constraint ;
```

```
                dc: title  "Grantee of use is exampleUniversity." ;
                schema:grantee       ex:exampleUniversity
            ]
        ]
    ] .

[   a sh:ValidationReport ;
    sh:conforms          true  ;
    sh: result           [
        a sh:ValidationResult  ;
        sh:focusNode         _:VerifiablePresentation  ;
        sh:sourceShape       ex:exampleShape#VerifiablePresentationShape ;
    ]
] .

_:VerifiablePresentation  odrl:hasPolicy ex:requestVP .

<#proofGraph> { ... }
```

4.5 Personal Data Usage Policy data model

In addition to agreeing on the use of *Resource R*, the client and server must also agree on the use of personal data from the VC. *Policy VP*, which is the subject of this section, is used for this purpose. Policy VP is a personal data usage agreement, therefore adheres to the OAC profile and is more comprehensive than a data usage agreement for non-personal data (*Policy R*), as personal data is a more valuable asset worth protecting. [18]

Personal data Usage Agreements are important to ensure the protection of personal data and to build trust between data owners and data users. At a time when personal information is a valuable resource, it is crucial to have clear agreements on how this data may be used. Personal Data Usage Agreements define the rules and restrictions that govern the access, processing and sharing of sensitive information. Such agreements not only protect the privacy of individuals, but also define the responsibilities of data users. They create transparency and give data holders control over the use of their personal data by enabling them to make informed decisions. Personal Data Usage Agreements thus help to strengthen the protection of personal data and raise awareness of the responsible use of sensitive information. [18] [16]

Policy R Request is a request that allows the server to read certain Verifiable Presentation attributes for identity verification based on legitimate interest. It ensures that access to the attributes complies with the defined purposes and legal requirements of the GDPR.

The example policy 15 provided below (*ex:requestVP*) is a request (*odrl:Request*) for access to Verifiable Presentation attributes for the purpose of identity verification based on a legitimate interest. The policy is created by *ex:exampleUniversity*.

The request contains a permission (*odrl:permission*) that defines *ex:exampleUniversity* as the assignee (*odrl:assignee*) for the action of reading (*oac:Read*) certain Verifiable Presentation attributes. The target of the permission is an RDF subject with the predicate *a* and the object *ex:Student*. These attributes are to be used to verify the identity of the client and are defined as required (*dc:isRequiredBy*) in the *ex:exampleShape#VerifiablePresentationShape*. The subject is modelled as blank node, since the server doesn't know the client's IRI yet.

The permission includes two constraints, which are linked by the use of *odrl:and*. The first constraint relates to the purpose (*dpv:Purpose*) of accessing the Verifiable Presentation attributes, which is to verify the identity of the permittee. The second restriction refers to the legal basis (*oac:LegalBasis*) for access, which is a legitimate interest (*dpv:LegitimateInterest*), since the server has a legitimate interest who accesses his data. The cause of defining a legal basis for the request to process personal data is explained in more detail in section 4.6.

Listing 15: Policy VP Request requests usage of VP attributes

```
ex:requestVP
    a odrl:Request ;
    odrl: profile                       oac: ;
    dc:description "Request to read Verifiable Presentation attribute for the
    purpose of identity verification based on legitimate interest." ;
    dc:creator                          ex:exampleUniversity ;
    dc:issued                           "2022-11-08T18:05:09"^^xsd:dateTime ;
    dc:valid                            "P1Y0M0D0H0S"^^xsd:duration ;
    dpv:hasDataProtectionOfficer        ex:exampleDataProtOfficer ;
    dpv:hasDataProcessor                ex:exampleEmployee ;
    odrl:uid                            ex:requestVP ;
    sec:proof                           <#proofGraph> ;
    odrl:permission      [
        odrl:assignee          ex:exampleUniversity ;
        odrl:action            oac:Read ;
        odrl:target            [
          rdf:subject              [ ] ;
          rdf:predicate            a ;
          rdf:object               ex:Student ;
          dc:isRequiredBy          ex:exampleShape#VerifiablePresentationShape ;
        ] ;
        odrl:constraint        [
          odrl:and
            [
              dc: title  "Purpose for access to Verifiable Credential attribute is
    to verify the identity of the assigner." ;
              odrl:leftOperand     dpv:Purpose ;
              odrl:operator        odrl:isA ;
              odrl:rightOperand    dpv: IdentityVerification
            ] ,
            [
              dc: title  "Legal basis for access is a legitimate interest." ;
              odrl:leftOperand     oac:LegalBasis ;
              odrl:operator        odrl:isA ;
              odrl:rightOperand    dpv:LegitimateInterest
            ]
        ]
    ] .

<#proofGraph> { ... }
```

After the client receives the request in Policy VP Request, he responds with the Policy VP Requirement in example 16 below. The content of these two documents is almost the same. The creator changes to *ex:exampleStudent* because the client creates the requirement. In addition, he adds the source of his personal data *ex:exampleStudent/vp_1*.

Listing 16: Policy VP Requirement - requires rules for usage of VP attributes

```
ex:requirementVP
    a oac:RequirementPolicy ;
    odrl: profile                    oac: ;
    dc:description "Requirement to read Verifiable Presentation attribute only for
    the purpose of identity verification based on legitimate interest." ;
    dc:creator                       ex:exampleStudent ;
    dc:issued                        "2022-11-08T18:05:09"^^xsd:dateTime ;
    dc:valid                         "P1Y0M0D0H0S"^^xsd:duration ;
    dc:references                    ex:requestVP ;
    dpv:hasDataProtectionOfficer     ex:exampleDataProtOfficer ;
    dpv:hasDataProcessor             ex:exampleEmployee ;
    odrl:uid                         ex:requirementVP ;
    sec:proof                        <#proofGraph> ;
    odrl:permission      [
        odrl:assigner        ex:exampleStudent ;
        odrl:action          oac:Read ;
        odrl:target          [
          rdf:subject            [ ] ;
          rdf:predicate          a ;
          rdf:object             ex:Student ;
          dc:isRequiredBy        ex:exampleShape#VerifiablePresentationShape ;
          dc:source              ex:exampleStudent/vp_1
        ] ;
        odrl:constraint      [
          odrl:and
            [
              dc: title "Purpose for access to Verifiable Credential attribute is
    to verify the identity of the assigner." ;
              odrl:leftOperand    dpv:Purpose ;
              odrl:operator       odrl:isA ;
              odrl:rightOperand   dpv: IdentityVerification
            ] ,
            [
              dc: title "Legal basis for access is legitimate interest." ;
              odrl:leftOperand    oac:LegalBasis ;
```

```
                odrl:operator        odrl:isA ;
                odrl:rightOperand    dpv:LegitimateInterest
            ]
        ]
    ] .

<#proofGraph> { ... }
```

After the server has compared his *Policy VP Request* with the *Policy VP Requirement* from the client and has determined that they match in content, he creates the *Policy VP Agreement* shown below (example 17). According to the OAC specification, he may only create an agreement if the two documents match [16]. The information contained remains the same, only the creator changes back to *ex:exampleUniversity*, the two previous documents are referenced and *ex:exampleStudent* is added as a data subject.

Finally, all necessary documents and policies are created to ensure a legally compliant exchange and processing of personal data. The digital signature in the *<#proofGraph>* allows both parties to prove later that an agreement on the terms of use has been reached.

Listing 17: Policy VP Agreement - agrees to rules for usage of VP attributes

```
ex:agreementVP
    a odrl:Agreement ;
    odrl: profile                       oac: ;
    dc:description "Agreement between an university and a student to read
    Verifiable Presentation attribute only for the purpose of identity
    verification based on legitimate interest." ;
    dc:creator                          ex:exampleUniversity ;
    dc:issued                           "2022-11-08T18:13:37"^^xsd:dateTime ;
    dc:valid                            "P1Y0M0D0H0S"^^xsd:duration ;
    dc:references                       ex:requirementVP, ex:requestVP ;
    dpv:hasDataProtectionOfficer        ex:exampleDataProtOfficer ;
    dpv:hasDataProcessor                ex:exampleEmployee ;
    odrl:uid                            ex:agreementVP ;
    sec:proof                           <#proofGraph> ;
    odrl:permission         [
        odrl:assigner           ex:exampleStudent ;
        odrl:assignee           ex:exampleUniversity ;
        odrl:action             oac:Read ;
        odrl:target             [
          rdf:subject             [ ] ;
          rdf:predicate           a ;
          rdf:object              ex:Student ;
```

```
        dc:isRequiredBy       ex:exampleShape#VerifiablePresentationShape ;
        dc:source             ex:exampleStudent/vp_1
      ] ;
      odrl:constraint [
        odrl:and
        [
          dc: title "Purpose for access to Verifiable Credential attribute is to
  verify the identity of the assigner." ;
          odrl:leftOperand    dpv:Purpose ;
          odrl:operator       odrl:isA ;
          odrl:rightOperand   dpv: IdentityVerification
        ] ,
        [
          dc: title "Legal basis for access is legitimate interest." ;
          odrl:leftOperand    oac:LegalBasis ;
          odrl:operator       odrl:isA ;
          odrl:rightOperand   dpv:LegitimateInterest
        ]
      ]
  ] .

<#proofGraph> { ... }
```

4.6 GDPR conformance of data models

The processing of personal data is strictly regulated by law in the EU. The General Data Protection Regulation (GDPR) came into power on 25 May 2018 and sets very high standards for privacy and security of data processing [18]. The GDPR obliges organizations to collect and process the personal data of EU citizens in a transparent and secure manner. This is not only to achieve security and privacy, but also to create a standard for processing. Violations of the GDPR are heavily penalized, with fines in the tens of millions of euros. In this section, the most important regulations of the GDPR are presented and their implementation in section 4.5 is explained.
The Personal Data Usage Policy defines terms of use for the Verifiable Presentation attributes of the client. Since the Verifiable Presentation is a tool to prove one's identity, the VP consists of personal data. The personal data is processed by the server, therefore the processing is regulated by the GDPR according to Article 2 of the GDPR.

The terms used in this thesis are defined in Art. 4 GDPR [18]:
Personal data refers to any information about a person (data subject) that can directly or indirectly identify them. This includes things like names, identification numbers, location data, online identifiers, and specific factors related to their physical, genetic, mental, economic, cultural, or social identity.
Processing of personal data covers any operation or set of operations performed on that data. It could e.g. involve collecting, recording, organizing, storing, or destroying the data, regardless of whether it is done manually or with automated tools.
Consent of the data subject means that the individual has freely, specifically, and informedly given their clear and unambiguous agreement or approval for their personal data to be processed in a certain way. This consent can be expressed through a statement or a clear affirmative action.

The GDPR states principles for the processing of data, listed below. Our goal is to model policies according to these principles and ensuring their implementation in the Verifiable Consent Protocol. However, there are still some issues that are not covered by our work so far, which is why we talk about aligning the data models with the GDPR and not about a full implementation of the regulations even if the wording suggests it. The principles of the GDPR therefore serve as a guide for the following modelling and may not be implemented in certain passages.

The principles of the GDPR for the processing of personal data derived from Art. 5 GDPR:

- **Lawfulness, fairness, and transparency:** Personal data must be processed in a way that is legal and open to the data subject. This means the processing must comply with applicable laws and regulations, treat individuals fairly, and be conducted

in a clear and transparent manner.

- **Purpose limitation:** Data should only be collected for specific, clear, and legitimate purposes. Once collected, it should not be used or processed in a way that is incompatible with these original purposes.
- **Data minimization:** The data collected should be sufficient, relevant, and limited to what is necessary for the purposes for which it is being processed. In other words, only the minimum amount of data required to achieve the specified purposes should be collected and used.

We implement the principle of lawfulness, fairness, and transparency by submitting a request for processing. This request defines which personal data are to be processed, who processes the data and on what legal basis the data may be processed. The client can therefore clearly see in advance what will happen to his personal data in the event of transmission to the server and can decide on the transmission on the basis of that information. If he agrees to the request, he responds with a condition on his part to the processing of the personal data. We assume that the server already makes a reasonable proposal and that the client therefore only has to agree and that the content of the request and the requirement is therefore the same. In any case, the server has received an explicit usage regulation from the client through the client's requirement and has already implicitly received consent through agreement on the content. It therefore accepts the requirement and sends an agreement to the client as an explicit confirmation of the agreement.
In addition, the principle of purpose limitation is addressed by the data models. Since a purpose must be specified for the processing according to Art. 13 c GDPR, the server must not deviate from this purpose. If he does, he is liable to prosecution and the client can use the digital signature to prove agreement for the original reason only.
The principle of data minimisation is implemented by the data model of the access control rule, since the SHACL shape only asks for data that serves the purpose of identity verification. On the client side, this minimisation must be implemented through selective disclosure mechanisms. However, if the server receives more information than it needs, it may not process it because it was not covered by a previous policy.

In order to further strengthen the rights of the data subject and to achieve transparency, the policies must contain the following information according to Art. 13 GDPR:

- the identity and the contact details of the controller
- the contact details of the data protection officer
- the purposes of the processing
- the legal basis of the processing

In our data model, this information is modelled with DPV terms and inserted at the appropriate positions in the OAC profile (see Listing 17).

The contact details are not included in the examples in section 4.5 due to space limitations, but must be communicated to the client as presented above. An exemplary modelling can be seen in the example Listing 18 below. These RDF graphs would need to be appended to the policies in section 4.5. The schema ontology provides standardized terms for the modelling of contact data.

Listing 18: Contact details example

```
ex:exampleDataProtOfficer foaf:name "John Muller" ;
  a dpv:DataProtectionOfficer ;
  schema:address [
    schema:addressLocality      "Anytown" ;
    schema:postalCode           "12345" ;
    schema:streetAddress        "123 Main Street" ;
    schema:addressCountry       "CountryX" ;
    schema:telephone            "012345" ;
    schema:email                "dataprotectionofficer@example.org"
  ] .

ex:exampleDataController foaf:name "University" ;
  a dpv:DataController ;
  a dpv:AcademicScientificorganisation ;
  schema:address [
    schema:addressLocality      "Anytown" ;
    schema:postalCode           "12345" ;
    schema:streetAddress        "123 Main Street" ;
    schema:addressCountry       "CountryX" ;
    schema:telephone            "012345" ;
    schema:email                "university@example.org"
  ] .
```

5 Verifiable Consent

In this chapter, we present a Verifiable Consent between a client and a server. Our objective is to develop a protocol that ensures that the given consent is cryptographically verifiable and transparent in the context of data usage. To achieve this, we will explain how to combine the modelled access control rules, non-personal data usage policies and personal data usage policies to a protocol. And we explain the Verifiable Consent Protocol in its whole procedure.

First, we introduce the Verifiable Consent Protocol in section 5.1, which combines the various components outlined in greater detail in the subsequent sections. This protocol aims to provide a mechanism for being able to state Access Control Rule (ACR)s based on Verifiable Credential (VC) attributes, validating them and ensuring individuals' consent in usage of that attributes.

Next, we focus on integrating non-personal data usage policies presented in section 4.4 in the protocol. These policies play a crucial role in determining how data can be utilized and for what purposes. By defining the data model for the policies in section 4.4 and utilizing it for the protocol, we aim to enable data provider to define more fine-grained terms of use for their data. Additionally, we delve into integrating personal data usage policies, presented in section 4.5. Privacy is a significant concern when it comes to data handling. By using personal data usage policies, we aim to incorporate measures that protect individuals' privacy rights and ensure their data is used in a manner that aligns with their expectations and preferences. We take the regulations of the GDPR as basis when modelling the personal data usage policies, to ensure a lawful data processing (see 4.6).

Throughout this chapter, we will address each step in detail, outlining the concepts, methodologies, and considerations involved in modelling a non-circumventable Verifiable Consent Protocol. By doing so, we hope to contribute to the development of decentralized authentication and authorization mechanisms and responsible data practices that respect individuals' rights and preferences.

5.1 Verifiable Consent Protocol

In this section, we present the Verifiable Consent Protocol. The aim of these explanations is to familiarize the reader with the protocol's process and to explain the relationships between the individual elements. In the following sections, we will look at the individual elements of the protocol in detail, so they are only mentioned in this section.

The accessing user is called the "client" in the following. On the semantic web, a client does not necessarily have to be a person, it may be an application or an organization. But in most cases, an individual acts on behalf of an organization or utilizes an application, so we assume the client to be a person in this work.

In the following, the term "server" refers to a solid server that provides the main functionality. This server stores and manages the provider's data and enables management via the Solid protocol in combination with the Verifiable Consent Protocol. The data structure of the Solid Protocol remains, while the WAC is supplemented by the VCP. Authentication is performed by the Verifiable Consent Protocol and authorization remains with the WAC. [14].

The Verifiable Consent Protocol describes the interaction between a server and a client. The goal of the protocol is to make the provider's *Resource R* stored on the server accessible to the client. All requests are made using HTTP and all documents used comply to the Resource Description Framework (RDF) [19][23].

Figure 1 gives an overview over the components of the protocol. On the left are all the documents that are created by the server. The documents on the right are those created by the client. The exception here is the Verifiable Credential (VC). It is created by a trusted third party and the client is the holder.He can dispose of it freely and creates the Verifiable Presentation (VP) from it independently. In the middle is the *Resource R*, for which the following protocol is used to regulate the access. *Policy R* defines terms of use for the *Resource R* and consists of an offer created by the server and an agreement created by the client. The client agrees to the offer by the server. The *Policy R* contains the *Access Rule for the Resource R*, which defines the Verifiable Presentation has to conform to the shape. The *Shape* is sent from the Server to the Client, in order to let the Client know, which proofs he has to present. That proofs are proofs of identity properties and therefore contain personal data. The terms of use for the personal data are defined in the *PolicyVP*. The server first creates a PolicyVP Request in order to inform the client about the intended usage. The client then responds to the request with the Policy VP Requirement, defining terms of use he permits or prohibits on his personal data. If the content of request and the requirement match, the Policy VP Agreement is created. The Agreement describes the consent on personal data usage, references the Request & the Requirement and contains a digital signature to prove its validity.

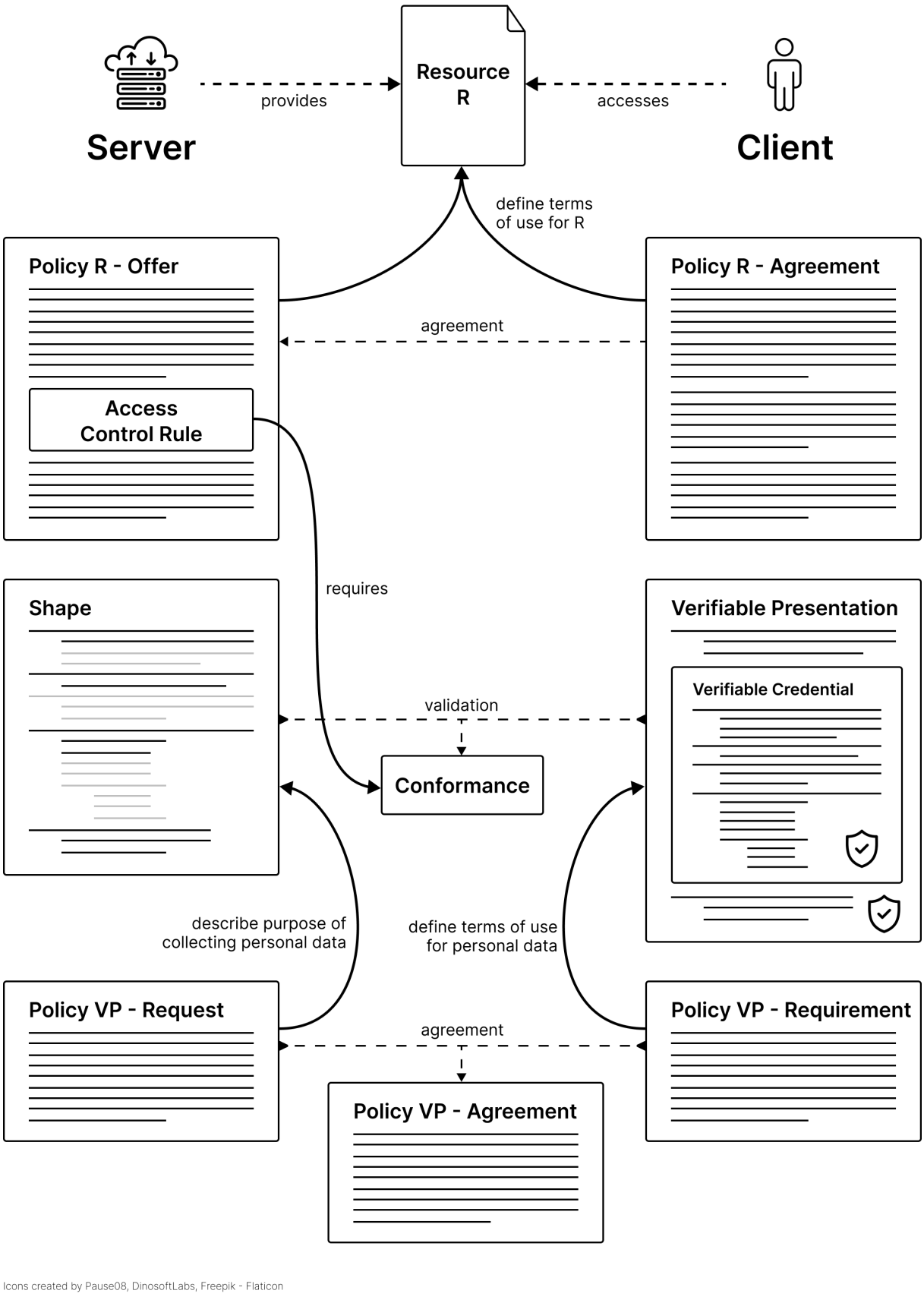

Figure 1: Overview over Verifiable Consent components

Figure 2 is a sequence diagram of the protocol and illustrates the sequence of HTTP requests. The client on the left makes the requests to the server on the right, which responds with a reply. The respective arrows are completed with the HTTP method used, the destination of the request and the information provided in the header.

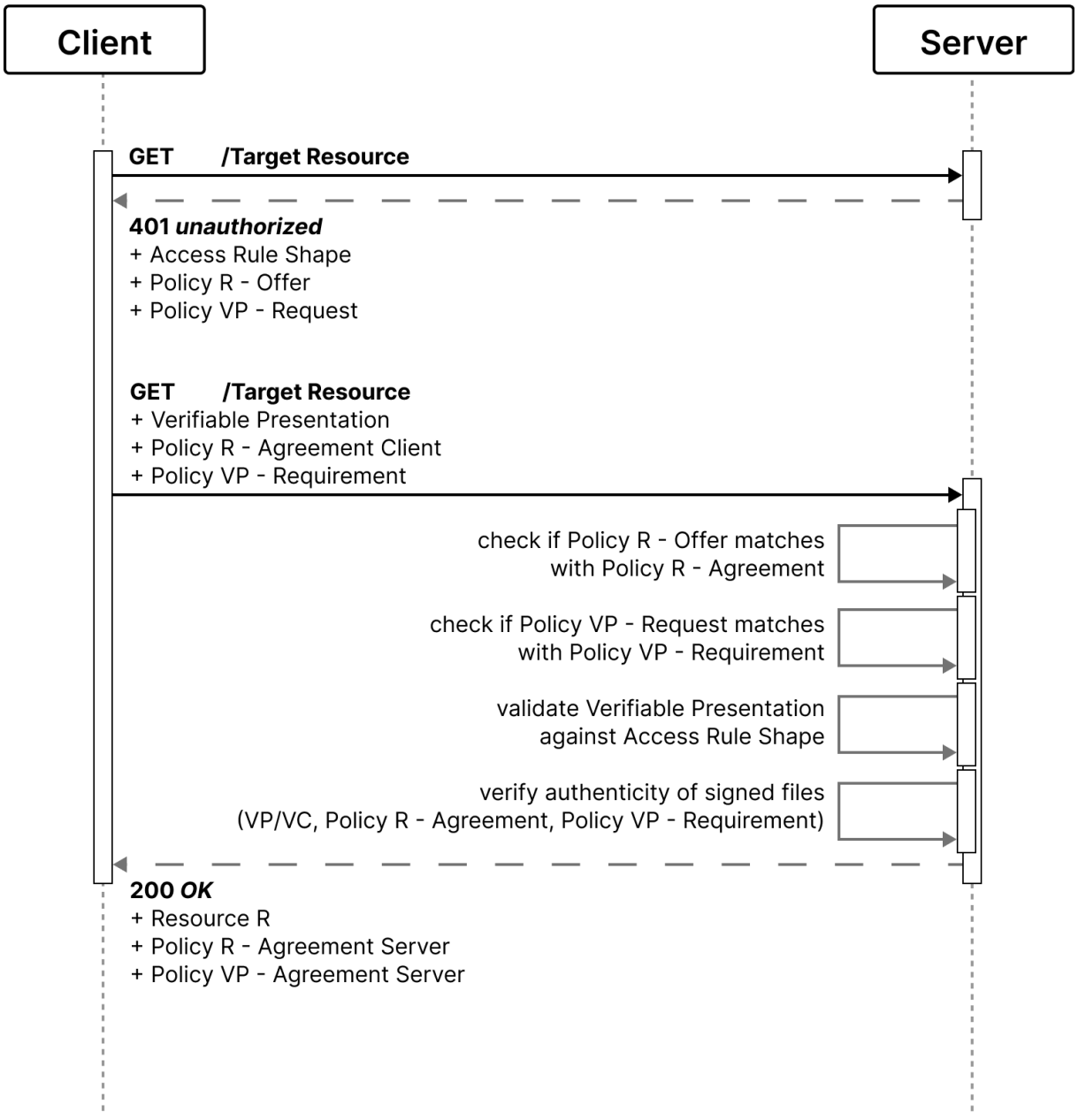

Figure 2: Sequence diagram of Verifiable Consent Protocol

First, the provider of the data defines the access rule. In order to do this, the provider creates an RDF graph shape that describes the required attributes for successful authentication and authorization. The provider stores the shape on its server and assigns the *Resource R* with an access rule. The access rule refers to the shape and access is only

granted if the shape is fulfilled by a presented RDF graph in the form of a Verifiable Presentation (VP). This step is only described for the sake of completeness. The actions in the paragraph above are not part of the protocol, since it requires an entity to create the shape. The protocol only contains actions of the client and the server.

Now the client tries to access *Resource R* via an HTTP GET method. The client does not provide anything in the header, because he doesn't know anything about the access rule yet, and therefore makes a simple request. The server responds with a *401 unauthorized* status code, so the client knows that he has been denied access [19]. In addition, the server provides the following documents within the HTTP header:

- The *Access Rule Shape* allowing the client to know which attributes to present in order to gain access.
- The *Policy R Offer*, which defines the usage rules for *Resource R* and refers to the *Access Rule Shape*.
- *Policy VP Request*, which provides context for processing the personal data to be presented from the VP. Here, usage rules are again defined for an RDF resource, but since this is personal data, *Policy VP* is much more comprehensive than *Policy R*.

The client now has all the information he needs to prepare for a new successful access attempt. According to the shape, the client can derive a suitable VP from its VC [29]. For example, selective disclosure as described in section 2.5 can be utilized here [9].

Policy R Offer presents the client with an offer to use *Resource R* for certain purposes, but also includes a duty to share its VP attributes to verify his identity. The client responds to this offer with the *Policy R Agreement*. The agreement should be consistent with the offer and restates all the rules from the offer. The creator of the offer is the server, while the creator of the agreement is the client. This way, a digital signature can be used to later prove that the client has agreed to the terms and conditions. If he did not agree to the server, he would unilaterally terminate the protocol, thereby gaining no disadvantage, since he has not yet shared any personal data with the server. However, he would also not gain access to the resource by terminating.

Policy VP Request contains information about the server's request to the client to be allowed to use the client's personal data for the purpose of identity verification. In this case though, the client does not accept this request with a direct agreement, as it concerns his personal data. Personal data is a legally higher valued than common data like *Resource R* and therefore needs an intermediate step during the agreement process. The client makes requirements for the processing of his personal data with the *Policy VP Requirement*, which the server has to accept in the next step. The content of the requirement must match the content of the request. The request therefore serves more as a proposal for

a regulation regarding the processing, which the client should adhere to if he wants to gain access. The client is the creator of the requirement and signs it digitally, so that the authenticity of the agreement between the two parties can be proven later.

The client now sends another GET request to the server to get the resource as a return. This time, however, the client delivers the VP, *Policy R Agreement* and *Policy VP Requirement*. This is followed by a comprehensive check of all information on the behalf of the server. It compares the content of the received *Policy R Agreement* with the preceding *Policy R Offer*. If they do not match, the server aborts the protocol, does not deliver *Resource R*, but is also not permitted to process the VP. The same procedure is used for *Policy VP*: The content of the received *Policy VP Requirement* is compared with the content of the sent *Policy VP Request*. According to the OAC definition, an agreement may only be concluded if there is a clear match [17]. The server then creates a *Policy VP Agreement*, which it later returns together with the *Resource R*. Furthermore, the server checks the access rule. For this purpose, the VP received is validated with the *Access Rule Shape*. Finally, if matching is successful, the signatures of all documents used are checked to verify their authenticity. Only through the use of digital signatures and the verification of these, a legally unobjectionable agreement can be proven later with certainty. For this purpose, a formal verification of the protocol is done in section 6.1. The order of the named checks is irrelevant for the theoretical Verifiable Consent Protocol, since the protocol would terminate with each unsuccessful check. To optimize the runtime of an implementation of the protocol, the complexity of the individual checks should be evaluated, and the order should be adjusted accordingly, in order to avoid unnecessary calculations. For example, a digital signature could not be valid, but according to 2 this is only checked at the very end of the checks. Before that, complex content comparisons of policies might be carried out, which are irrelevant because the protocol terminates anyway due to the invalid signature.

As a final step, the server responds to the client's request with *Resource R* after successfully matching all conditions. The client can now process the information for the purpose defined in *Policy R*. To ensure that no information regarding the terms of use for the resource is lost, the *Policy R Agreement* is returned again. This time not only with the signature of the client, but also with the signature of the server. In addition, the created *Policy VP Agreement* is shared with the client. This way, the client also has all the necessary policy agreements to prove an agreement later on.

For each resource, this protocol must be followed and run through again.

6 Evaluation

In this section, we evaluate the Verifiable Consent Protocol regarding its ability to withstand attacks from third parties and dishonest agents. To make sure the Verifiable Consent Protocol 5.1 works well and keeps data safe, it needs to be cryptographically evaluated. In this evaluation, we use a tool called ProVerif [6] to check how the Verifiable Consent Protocol is prone to attacks. Formal verification methods like ProVerif are suitable for analysing the protocol's security, finding any weaknesses, and making sure it meets the desired requirements listed in section 6.1.

Formal verification is an approach in computer science and software engineering that aims to mathematically prove the correctness and reliability of systems, programs or hardware designs. It is a formal and mathematical process that uses techniques of formal logic and formal languages to ensure that a system or programme meets its specified requirements and safety guarantees. [6]

The main goal of this evaluation is to fully understand how effective the Verifiable Consent Protocol is in protecting data privacy, integrity, confidentiality and enabling non-repudiation. During this evaluation, we'll look at different parts of the Verifiable Consent Protocol, like how it communicates, authenticates users, signs information and encrypts information. By examining these aspects, we can see how well the protocol prevents unauthorized access, data leaks, and other security issues. Using ProVerif, we can find any technical problems or weaknesses that might exist and give recommendations for improvement.

6.1 Formal Verification of Verifiable Consent Protocol

In this section, we evaluate the Verifiable Consent Protocol for its ability to be compromised by dishonest parties or an attacker. For this test, we use the software tool ProVerif. ProVerif is a tool used in cryptography and security research to perform formal verification of security protocols. It was developed by Bruno Blanchet and is particularly useful for verifying the security properties of cryptographic protocols like the Verifiable Consent Protocol. The tool is based on the formal "Dolev-Yao model" and works with a special logical language called "Applied Pi-Calculus". With ProVerif, protocol sequences can be modelled and then automatically checked for possible security issues and vulnerabilities. The result of an analysis by ProVerif consists of a simple "True" or "False" depending on whether the checked part of the protocol meets the specified security properties or not. If the tool finds a vulnerability, a corresponding detailed description of the successful attack is provided. [6]

The following subsections present different requirements for the Formal Verification. First *Secrecy & Confidentiality* is presented in section 6.1.1, then *Reliability* in section 6.1.2 and finally *Non-repudiation* in section 6.1.3.

In the following sections, excerpts of the ProVerif code are presented, and their functionality is explained. A combined version of the code can be found in the Appendix or on GitHub.

6.1.1 Secrecy & Confidentiality

For the Verifiable Consent Protocol, one desired property is particularly important: secrecy. No third party should be able to intercept messages between an honest client and an honest server to obtain information that is not supposed to be used by him. ProVerif can check whether the Verifiable Consent Protocol fulfils the secrecy property and thus does not permit a data breach. To ensure secrecy, the client and server must cryptographically encrypt the messages they exchange. In this thesis, we have chosen a symmetrical encryption method. Symmetric encryption is characterized by creating a shared key at the beginning of a contact between server and client, which is then used to encrypt the subsequent messages. In contrast to asymmetric encryption, both parties use the same key here. For this initial agreement on a common key, we use a Diffie-Hellman key exchange. The Diffie-Hellman key exchange allows an interception-secure agreement on a shared symmetric key and is often used for symmetric encryption methods. With a Diffie-Hellman key exchange, a key can be shared via a public channel that is only known to the two parties and cannot be calculated by third parties. [22] In an asymmetric encryption method, the server would always encrypt messages for the client with the client's public key, and the client would decrypt the message with its private key. To reply to the server,

the client would then encrypt the message with the server's public key and the server can decrypt the message with its private key. [28]

The property confidentiality is ensured by using a digital signature for message contents. The digital signature can ensure the integrity of the messages. The signature is created by a trusted entity (usually a party with a private key) and attached to the message. When the message arrives, the recipient can use the sender's public key to verify the signature to ensure that the message has not been manipulated and is indeed from the expected sender. If the signature verification is successful, the recipient can be sure that the message has not been changed during transmission. [28]

The ProVerif representation in Listing 19 begins with the definitions of the encryption and digital signature. The upper section lines 1-6 declare the data types of the different keys, that are later utilized by the server and the client. The second section lines 8-17 declares needed data types, functions and equations for the Diffie-Hellman key exchange. In lines 19-22, the declaration of the symmetric encryption is contained. First a constructor for encrypting messages is defined, thereafter a deconstructor to decrypt messages is declared. The encryption function demands a message and a symmetric key as input, and returns an encrypted message. The decryption function subsequently returns the message content when an encrypted message and the same symmetric key are the inputs. The last section in lines 64-67 declares functions to apply an asymmetric digital signature to messages. The function declared in line 25 signs a message with a secret key and returns a signature value. The deconstructor in line 26 can validate whether a signed message was not changed. Therefore, the *check* function takes the message to validate, its signature value and the public key of the party which signed the message as an input. If the calculated value of the public key and the send signature value match, the message is not edited. The receiver of the message can therefore check the integrity of the message without the help of a third party. [7][28]

Listing 19: Encryption & Signature - Excerpt from ProVerif representation of Verifiable Consent Protocol

```
1  (* START keys *)
2  type SecretKey.
3  type PublicKey.
4  type SymmetricKey.
5  fun pk(SecretKey):PublicKey.
6  (* END keys *)
7
8  (* START group *)
9  type G.
10 type exponent.
```

```
11 fun exp(G, exponent): G.
12 fun g(exponent): G.
13 equation forall x: exponent, y: exponent; exp(g(x),y) = exp(g(y),x).
14 fun GtoSymK(G): SymmetricKey.
15 fun bitG(G):bitstring.
16 reduc forall elem:G; unbitG(bitG(elem))=elem.
17 (* END group *)
18
19 (* START symmetric encryption *)
20 fun enc(bitstring(*the message*), SymmetricKey):bitstring (*symmetric encrypt input msg*).
21 reduc forall msg:bitstring,symk:SymmetricKey; dec(enc(msg,symk),symk)=msg (*symmetric
       decrypt*).
22 (* END symmetric encryption *)
23
24 (* START digital signature *)
25 fun sig( bitstring (*the message*), SecretKey):bitstring(*the signature value*).
26 reduc forall msg:bitstring,sk:SecretKey; check(msg,sig(msg,sk),pk(sk))=true
       (*check(msg,sigVal,pk)*).
27 (* END digital signature *)
```

After the declarations, we will explain the process macros regarding secrecy in the following. In the combined ProVerif code in the Appendix A.1, the protocol from the client's perspective is defined first and the perspective of the server is defined after. In the following we will alternate between the perspectives as the sequence of the protocol is simpler to follow then.

Listing 20: The process macros of the protocol start with the definition of the client. As seen in Listing 20 in line 1, the client has a secret key and knows the public key of the server. Both variables are private and only known by the client. The sequence of the protocol starts with a handshake-request utilizing the Diffie-Hellman key exchange. The first message *m_ 0* is sent via the channel *h*.

Listing 20: ProVerif excerpt - secrecy 1

```
1 let client (sk_c:SecretKey, pk_s:PublicKey, talksOnlyToHonest:bool) (* input ~ what the
      agent starts with in private*) =
2     new x:exponent;
3     let gx = g(x) in
4     let m'_0 = bitG(gx) in
5     let m_0 = m'_0 in
6     out(h, m_0);
7     (* Sent out Handshake-Request *)
```

Listing 21: The server then receives the handshake request, executes his part of the handshake actions, calculates the symmetric key K and responds with *m_1*. The server also has a secret key and the public key of the client, which is defined in line 1.

Listing 21: ProVerif excerpt - secrecy 2

```
1  let server(sk_s:SecretKey, pk_c:PublicKey, talksOnlyToHonest:bool) (* input ~ what the
       agent starts with in private*) =
2      (* Receive Handshake-Request *)
3      in(h, m_0:bitstring);
4      let gx= unbitG(m_0) in
5      new y:exponent;
6      let gy = g(y) in
7      let K = GtoSymK(exp(gx,y)) in
8      let m'_1 = sig((gy,gx),sk_s) in
9      let m_1 = (bitG(gy),enc(m'_1, K)) in
10     out(h,m_1);
11     (* Sent out Handshake-Response *)
```

Listing 22: *m_1* is received by the client. He calculates the symmetric key K and sends out a data request to the target resource located represented by *m_uri*. m_uri describes the address of the resource. The address is necessary because individual requests are independent of each other. To ensure that the target of the access does not change in the course of the protocol, the requests are linked with the help of the m_uri variable. This is a simplification made for this thesis. In reality, a session token or similar would be used here.

Listing 22: ProVerif excerpt - secrecy 3

```
1      (* Receive Handshake-Response *)
2      in(h,m_1:bitstring);
3      let (eGY:bitstring,m:bitstring) = m_1 in
4      let gy = unbitG(eGY) in
5      let K = GtoSymK(exp(gy,x)) in
6      let s_K = dec(m,K) in
7      if check((gy,gx), s_K, pk_s) then
8      let m'_2 = (sig((gx,gy), sk_c), m_uri) in
9      let m_2 = enc(m'_2, K) in
10     out(h,m_2);
11     (* Sent out data request *)
```

The two Listings 21 & 22 above show an example of how symmetrical encryption and digital signature work. First the digital signature: The content *(gy,gx)* of the message

m'_1 is signed in 21 in line 8 with the secret key of the server: *sig((gy,gx),sk_s)*. The message is then sent to the server and gets checked in line 7 of Listing 22: *check((gy,gx), s_K, pk_s)*. The hash value of the message content *(gx,gy)* calculated with the server's public key is compared to the signature value *s_K*. If these values match, the returned result of the *check* function is the boolean *true*.

The symmetric encryption: The *enc* function (21 line 9) is used for encrypting a message using the symmetric key *K*, while the *dec* function (22 line 6) is used to decrypt the message using the same symmetric key. That principle is used for every message that is sent and received in the sequence.

The combination of a digital signature and an encryption in one protocol thus ensures that messages not only originate from the correct source (integrity), but can also only be read by authorized parties (confidentiality). This is particularly important when confidential information is transmitted over public networks, as the signature ensures that the message has not been manipulated by an attacker and the encryption ensures that the information remains secret when transmitted over the network.

6.1.2 Reliability

We are convinced that reliability is an important property for a protocol that describes the handling of personal data, as reliability strengthens the trust of users in the protocol. The Verifiable Consent Protocol is intended to implement the guidelines of data protection law as well as possible and should therefore not have a security gap. A security gap in the course of the agreement would allow unauthorized access or unauthorized processing of personal data. The implementation of the protocol must therefore ensure that data is adequately protected, that the processing of the data is reliably regulated and that access is only possible for authorized actors. A reliable protocol also ensures the availability of data. Only if the procedures of a protocol are clearly regulated and cannot be deviated from, a trustworthy exchange of data possible is for all actors at all times. Users who want to access data can do so at any time as once they have presented the required proof of identity. On the other hand, the provider of the data is sure that his data is available at all times and only allows authorized access. Furthermore, the provider of the data is certain that it will not face any legal consequences, as the data processing is reliably communicated and carried out. Many countries and regions have strict data protection laws and regulations that govern the handling of personal data. A reliable protocol facilitates compliance with these regulations and minimizes the risk of fines or legal consequences due to violations.

In the following section, we will test the flow of the protocol with ProVerif for security vulnerabilities and aim to prove the reliability of the Verifiable Consent Protocol. To do

this, variables and events must first be declared in Listing 23, with which the flow of the protocol can be modelled. In the block "Communication" in lines 74-78, the communication channel between server and client is declared and the contents of the messages. *Offer, Request* and *Shape* are documents that the server provides to the client after its first request (see sequence diagram 2).

In the course of the protocol, an equality is defined between the versions of Policy R (R_offer & R_agreement) and Policy VP (VP_request, VP_requirement & VP_agreement) respectively. This is to represent the required content match of the policy version. Again, this is simplified by strict equality; in reality, a detailed evaluation of the content match of the RDF documents would have to be done.

Furthermore, another channel and a private resource are defined in lines 8-9. These declarations have the benefit in the course of verification that it can be checked whether an attacker was able to eavesdrop on the resource during a communication between an honest client and an honest server. A more detailed explanation follows in the course of this section. In the "Protocol Completes" block in lines 11-13, functional variables are declared. With the help of these, it can later be checked whether the protocol can be successfully completed at all.

Afterwards, from line 17 to line 20, a series of events is declared, which are inserted at the positions in the protocol in order to determine that e.g. the resource has been sent to the client. In the block "Sharing Events", states of the data exchange are described step by step from bottom to top. The client sends the credential, the server receives it, the server then sends the resource and the client receives it. Accordingly, the presentation is listed as the argument of the lower two events. For the upper two events, the resource is listed. In addition, a secret key is defined as an argument in each of the events. Only an honest actor has the secret key of an honest actor. With the secret key, it can be checked whether it is really the client or server that has received or sent the resource (or the presentation).

This is followed by the declarations of the states of the agreement in block agreement events lines 22-24. Here, too, a state is defined step by step from bottom to top. With the help of the demand of the secret key, alike above, it can be determined whether the actor really has the correct identity and is not a dishonest actor. The server first proposes rules for the processing of data and the client accepts this offer. At this point, there is an implicit consent about the terms for data processing because the client has accepted the server's offer. As soon as the server has also agreed, there is an explicit consent.

The events are placed at the corresponding positions in the protocol flow and thus provide an abstract representation of states at which the properties defined in Listing 23 are met.

Listing 23: ProVerif excerpt - reliability definitions

```
(* Communication *)
free h:channel.
free m_uri:bitstring.
free m_R_offer:bitstring.
free m_VP_request:bitstring.
free m_shape:bitstring.

(* Interception Check *)
fun ch(bitstring): channel.
free resource_is_private: bitstring [private].

(* Protocol Completes / Sanity checks *)
free Client_completes:bitstring [private].
free Server_completes:bitstring [private].

(* Sharing Events *)
event Client_received_resource(SecretKey, bitstring (* sk proves identity, m_resource *)).
event Server_provided_resource(SecretKey, bitstring (* sk proves identity, m_resource *)).
event Client_provided_credential(SecretKey, bitstring (* sk proves identity,
    m_presentation *)).
event Server_received_credential(SecretKey, bitstring (* sk proves identity,
    m_presentation *)).

(* Agreement Events *)
event Server_has_agreed(SecretKey, bitstring,bitstring (* sk_s, m_R_agreement,
    m_VP_agreement *)).
event Client_has_agreed(SecretKey, bitstring,bitstring (* sk_c, m_R_agreement,
    m_VP_requirement *)).
event Server_has_offered(SecretKey, bitstring,bitstring, bitstring (* sk_s, m_shape,
    m_R_offer, m_VP_request *)).
```

The defined events can be linked by queries. Thus, it can be modelled if event b was executed, event a had to be executed before. If the result of the query is "true" after the verification with ProVerif, the sequence $a => b$ is also true and there is no attack possible. With this functionality, we can prove the reliability of the Verifiable Consent Protocol.

Listing 24 represents the query that specifies the sequence of "Agreement" and "Sharing" events from Listing 23. The sequence consists of nested correspondences. The event *server has offered* must be executed first, and the event *client received resource* last, so that the query's result is true. *inj-event* indicates an injective correspondence is desired. The Verifiable Consent protocol must be completed for each new data request, so an injective relationship exists between the events in the ProVerif code. This means that there is a one-to-one relationship between each event in the query. So for each resource received, the Server must make an offer. [7]

Listing 24: ProVerif excerpt - reliability query

```
query sk_c:SecretKey, sk_s:SecretKey, credential:bitstring, policyR:bitstring,
    policyVP:bitstring, resource:bitstring;
    inj-event(Client_received_resource(sk_c, resource))
    ==>
        ( inj-event(Server_provided_resource(sk_s, resource))
        ==>
            ( inj-event(Server_received_credential(sk_s, credential))
            ==>
                ( inj-event(Client_provided_credential(sk_c, credential))
                ==>
                    ( inj-event(Server_has_agreed(sk_s, policyR, policyVP))
                    ==>
                        ( inj-event(Client_has_agreed(sk_c, policyR, policyVP))
                        ==>
                        inj-event(Server_has_offered(sk_s, credential, policyR, policyVP))
                        )
                    )
                )
            )
        )
    .
```

The result of the Query above is shown below in Listing 25. With the result being true, we prove there was no attack against the query discovered. Respectively, the flow of the Verifiable Consent Protocol is not attackable by a dishonest actor who wants to circumvent the protocol in order to get the resource.

Listing 25: ProVerif excerpt - reliability query result

```
Query inj-event(Client_received_resource(sk_c_3, resource))
  ==> (inj-event(Server_provided_resource(sk_s_3, resource))
  ==> (inj-event(Server_received_credential(sk_s_3, credential))
  ==> (inj-event(Client_provided_credential(sk_c_3, credential))
  ==> (inj-event(Server_has_agreed(sk_s_3, policyR, policyVP))
  ==> (inj-event(Client_has_agreed(sk_c_3, policyR, policyVP))
  ==> inj-event(Server_has_offered(sk_s_3, credential, policyR, policyVP)
  ))))))  is  true.
```

6.1.3 Non-repudiation

The third important property of the protocol is “non-repudiation”. Non-repudiation is an essential aspect of information security in information technology systems. It refers to the ability to prove the origin and integrity of digital transactions or communications so that a party involved is not later able to deny its involvement or statements. Non-repudiation is critical to ensure trust in digital business processes, electronic transactions and legal agreements. The ISO (International Organization for Standardization) has developed guidelines and standards to support the implementation of non-repudiation in information technology systems [1].

According to ISO [1] the definition of non-repudation is as follows: *“The goal of the Non-repudiation service is to collect, maintain, make available and validate irrefutable evidence concerning a claimed event or action in order to resolve disputes about the occurrence or non-occurrence of the event or action.” [1]*

In this thesis, the Verifiable Consent Protocol ensures that each party can prove consent. In the following, we want to use ProVerif to evaluate whether the non-repudiation property is vulnerable. For this purpose, the associated events are defined in Listing 26. Here, it is modelled from the perspective of the client and the server with which information it can be proven that the server has made an offer, the client has agreed, and finally the server has agreed. If this sequence can be proven by both agents, no deniability by the other party is possible afterwards. The proof of an assertion is ensured by a digital signature. If, for example, the client only has the contents of the shape, the R offer and the VP request, he cannot prove with certainty that the contents have not been changed. Only by digitally signing the contents the client can prove the authenticity of the shape, the offer and the request.

Listing 26: Non-repudiation events - Excerpt from ProVerif representation of Verifiable Consent Protocol

```
(* Non-Repudiation Events *)
event Client_can_prove_server_has_offered(bitstring, bitstring, bitstring, bitstring,
    bitstring, bitstring (* m_shape, sig_shape, m_R_offer, sig_R_offer, m_VP_request,
    sig_VP_request *)).
event Client_can_prove_client_has_agreed(bitstring, bitstring, bitstring, bitstring (*
    m_R_agreement, sig_R_agreement_client, m_VP_agreement, sig_VP_agreement *)).
event Client_can_prove_server_has_agreed(bitstring, bitstring, bitstring, bitstring (*
    m_R_agreement, sig_R_agreement_server, m_VP_agreement, sig_VP_agreement *)).
event Server_can_prove_server_has_offered(bitstring, bitstring, bitstring, bitstring,
    bitstring, bitstring (* m_shape, sig_shape, m_R_offer, sig_R_offer, m_VP_request,
    sig_VP_request *)).
event Server_can_prove_client_has_agreed(bitstring, bitstring, bitstring, bitstring (*
    m_R_agreement, sig_R_agreement_client, m_VP_requirement, sig_VP_requirement
    *)).
event Server_can_prove_server_has_agreed(bitstring, bitstring, bitstring, bitstring (*
    m_R_agreement, sig_R_agreement_server, m_VP_agreement, sig_VP_agreement *)).
```

In Listing 27 below, the "non-repudation events" from above are queried. If the client can prove the server has agreed, the server had to agree before. By requiring the secret key of the server, its identity is proven. We check all “Agreement Events” from 23 to see if both parties can prove this agreement event by having seen the signatures.

Listing 27: Query non-repudiation events - Excerpt from ProVerif representation of Verifiable Consent Protocol

```
query sk_s:SecretKey, policyVP:bitstring, policyR:bitstring, sig_policyR:bitstring,
    sig_policyVP:bitstring;
    inj-event(Client_can_prove_server_has_agreed(policyR, sig_policyR, policyVP,
    sig_policyVP))
    ==>
    inj-event(Server_has_agreed(sk_s, policyR, policyVP))
    .

query sk_c:SecretKey, policyVP:bitstring, policyR:bitstring, sig_policyR:bitstring,
    sig_policyVP:bitstring;
    inj-event(Server_can_prove_client_has_agreed(policyR, sig_policyR, policyVP,
    sig_policyVP))
    ==>
    inj-event(Client_has_agreed(sk_c, policyR, policyVP))
    .
```

```
12
13 query sk_s:SecretKey, policyVP:bitstring, policyR:bitstring, sig_policyR:bitstring,
       sig_policyVP:bitstring, credential:bitstring, sig_credential:bitstring;
14     inj-event(Client_can_prove_server_has_offered(credential, sig_credential, policyR,
       sig_policyR, policyVP, sig_policyVP))
15     ==>
16     inj-event(Server_has_offered(sk_s, credential, policyR, policyVP))
17     .
18
19 query sk_s:SecretKey, policyVP:bitstring, policyR:bitstring, sig_policyR:bitstring,
       sig_policyVP:bitstring, credential:bitstring, sig_credential:bitstring;
20     inj-event(Server_can_prove_server_has_offered(credential, sig_credential, policyR,
       sig_policyR, policyVP, sig_policyVP))
21     ==>
22     inj-event(Server_has_offered(sk_s, credential, policyR, policyVP))
23     .
24
25 query sk_s:SecretKey, policyVP:bitstring, policyR:bitstring, sig_policyR:bitstring,
       sig_policyVP:bitstring;
26     inj-event(Server_can_prove_server_has_agreed(policyR, sig_policyR, policyVP,
       sig_policyVP))
27     ==>
28     inj-event(Server_has_agreed(sk_s, policyR, policyVP))
29     .
30
31 query sk_c:SecretKey, policyVP:bitstring, policyR:bitstring, sig_policyR:bitstring,
       sig_policyVP:bitstring;
32     inj-event(Client_can_prove_client_has_agreed(policyR, sig_policyR, policyVP,
       sig_policyVP))
33     ==>
34     inj-event(Client_has_agreed(sk_c, policyR, policyVP))
35     .
```

The non-repudiation framework of ISO distinguishes between two different types of non-repudiation:

1.) Non-repudiation with proof of origin: A sender of a message should not be able to deny a statement he made earlier [1]. A system that wants to implement non-repudiation should therefore be able to prove which content, which user, at what time, has sent to whom. In this thesis, the system is decentralized. It follows that no third party can prove the communication between two agents. Therefore, the Verifiable Consent Protocol must ensure that the two agents can present non-repudiable evidence. The evidence

must contain there was an agreement and the content of the agreement. With that, the decentralized agreement could later be proven to a third party, the court, for example. Since the agreement only takes place between the two agents, it must be ensured that the content of the two pieces of evidence cannot be different in any case. The ProVerif result in Listing 28 proves that the Verifiable Consent Protocol only creates non-repudiable agreements because all queries are true.

Listing 28: Result non-repudiation queries - Excerpt from ProVerif result of Verifiable Consent Protocol

```
1  Query inj-event(Client_can_prove_server_has_agreed(policyR, sig_policyR, policyVP,
       sig_policyVP))
2    ==> inj-event(Server_has_agreed(sk_s_3, policyR, policyVP)) is true.
3
4  Query inj-event(Server_can_prove_client_has_agreed(policyR, sig_policyR, policyVP,
       sig_policyVP))
5    ==> inj-event(Client_has_agreed(sk_c_3, policyR, policyVP)) is true.
6
7  Query inj-event(Client_can_prove_server_has_offered(credential, sig_credential, policyR,
       sig_policyR, policyVP, sig_policyVP))
8    ==> inj-event(Server_has_offered(sk_s_3, credential, policyR, policyVP)) is true.
9
10 Query inj-event(Server_can_prove_server_has_offered(credential, sig_credential, policyR,
       sig_policyR, policyVP, sig_policyVP))
11   ==> inj-event(Server_has_offered(sk_s_3, credential, policyR, policyVP)) is true.
12
13 Query inj-event(Server_can_prove_server_has_agreed(policyR, sig_policyR, policyVP,
       sig_policyVP))
14   ==> inj-event(Server_has_agreed(sk_s_3, policyR, policyVP)) is true.
15
16 Query inj-event(Client_can_prove_client_has_agreed(policyR, sig_policyR, policyVP,
       sig_policyVP))
17   ==> inj-event(Client_has_agreed(sk_c_3, policyR, policyVP)) is true.
```

Additionally, to ensure non-repudiation with proof of origin even further, the protocol could be extended by an additional check of the server side's identity. The data provider could provide the client with a Verifiable Presentation in order to prove the references made in the policies regarding the data provider's person. Without the proof of identity, the server could be dishonest in the way that he agrees to the terms of data usage under a false identity.

2.) Non-repudiation with proof of delivery: An agent should be able to prove the agreement reached the right recipient [1]. In this thesis, the second type of non-repudiation

is evaluated by proving with ProVerif, the client has an explicit signed agreement from the server and the server has a signed agreement from the client. As the data model requires an assignee and an assigner, the proof of delivery is possible. The Policies are explicitly meant for a client after he agrees to the terms of use. Before there is no explicit mentioning of the client possible as the server does not know him yet. But that does not matter for the non-repudiation property because there is no agreement yet.
Additionally, the policies are exchanged not only once between the agents. Every state of a policy references the previous state and therefore is linked to the old message. If the client signs the *Agreement R* and *Requirement VP*, and sends it to the server, the server can prove, the *Offer R* and *Request VP* reached the right recipient. Likewise, the server sends signed *Agreements R and VP* to the client, so the client has a proof, *Agreement R* and *Requirement VP* reached the right recipient. The only action that can not be proven is the transmission of both Agreements from server to client, because the protocol ends after that action. But with the transmission of the *Offer R* and *Request VP*, the server can at least prove the delivery of an implicit consent between the two parties.

To completely satisfy the need for "Non-repudiation with proof of delivery", we propose a adjustment of the protocol as follows:
We add an extra step in Figure 3, which ensures a transmission of signed agreements from the client to the server before the resource is transmitted. With that adjustment, we would further ensure non-repudiation because both parties sign the agreements. The server creates them, signs them and transmits them to the client. He then accesses them via a GET method, signs the agreements inlcuding the server's signature, and returns them to the server. Additionally, this extended form would allow explicit consent before personal data is shared. The client no longer has to transmit personal data based on an implicit consent, but can already prove that an explicit agreement exists. For easier handling, however, we decided against the extended form in this thesis, since an implicit consent is provable. The additional assurance also requires one more request and is therefore too extensive in our opinion.

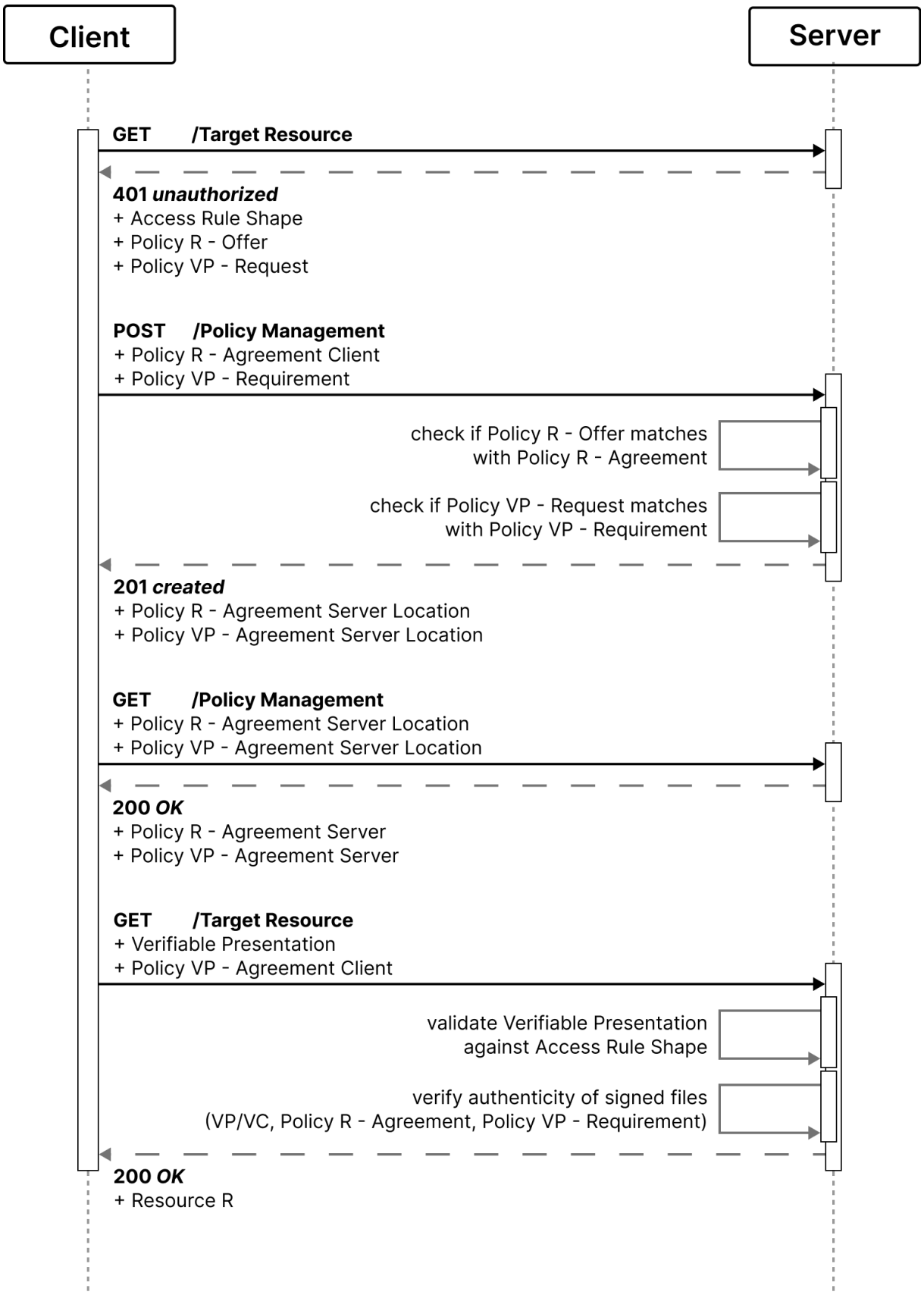

Figure 3: Sequence diagram of adjusted Verifiable Consent Protocol

6.2 Conclusion

The Verifiable Consent Protocol enables a GDPR-compliant exchange of decentralized data without a central authentication server. The central server is not needed because only decentralized technologies such as the Verifiable Credential Model and the Solid Protocol are used. To conclude this thesis, we will answer the research questions asked at the beginning of the thesis:

(a) How can requirements for authentication and authorization be modelled in a way that is interoperable with existing approaches to model Verifiable Credentials?
We have developed a data model for the access control system, ensuring that no unauthorized user can access data. At the same time, the data model enables easy modelling of the access mode for authorised users whose Verifiable Presentation fulfils the conditions of the SHACL Shape.

(b) How can all parties be aware of all terms related to the processing of (personal) data and be able to prove a reached agreement?
Through the combination of non-personal data usage policies and personal data usage policies, which are created before the resource is handed over, the exchange of private data is legally protected for both the client and the server. The client receives transparent information about how his data is processed and can prove which purposes he has expressly agreed to or prohibited. The server, on the other hand, is protected if the client tries to take legal action against it, as it too has acted in a fully traceable GDPR-compliant manner and can prove the client's consent. The consent described by the policies cannot be denied. We were able to demonstrate the proneness of the protocol against dishonest agents and man-in-the-middle attacks through a formal verification.

A Appendix

Policy data models, ProVerif code and a complete version of the ProVerif result can be found on GitHub.

A.1 ProVerif code - Verifiable Consent Protocol

Listing 29: ProVerif description of the Protocol Flow from the Verifiable Consent Protocol

```
(* This is a ProVerif description of the Protocol Flow from the Verifiable Consent
    Protocol. *)

(* START keys *)
type SecretKey.
type PublicKey.
type SymmetricKey.
fun pk(SecretKey):PublicKey.
(* END keys *)

(* START group *)
type G.
type exponent.
fun exp(G, exponent): G.
fun g(exponent): G.
equation forall x: exponent, y: exponent; exp(g(x),y) = exp(g(y),x).
fun GtoSymK(G): SymmetricKey.
fun bitG(G):bitstring.
reduc forall elem:G; unbitG(bitG(elem))=elem.
(* END group *)

(* START symmetric encryption *)
fun enc(bitstring(*the message*), SymmetricKey):bitstring (*symmetric encrypt input msg*).
reduc forall msg:bitstring,symk:SymmetricKey; dec(enc(msg,symk),symk)=msg (*symmetric
    decrypt*).
(* END symmetric encryption *)

(* START digital signature *)
fun sig( bitstring (*the message*), SecretKey):bitstring(*the signature value*).
reduc forall msg:bitstring,sk:SecretKey; check(msg,sig(msg,sk),pk(sk))=true
    (*check(msg,sigVal,pk)*).
(* END digital signature *)
```

```
30
31 (* DEFS *)
32
33 (* Communication *)
34 free h:channel.
35 free m_uri:bitstring.
36 free m_R_offer:bitstring.
37 free m_VP_request:bitstring.
38 free m_shape:bitstring.
39
40 fun ch(bitstring): channel.
41 free resource_is_private: bitstring [private].
42
43 (* Protocol Completes / Sanity checks *)
44 free Client_completes:bitstring [private].
45 free Server_completes:bitstring [private].
46
47 (* Sharing Events *)
48 event Client_received_resource(SecretKey, bitstring (* sk proves identity, m_resource *)).
49 event Server_provided_resource(SecretKey, bitstring (* sk proves identity, m_resource *)).
50 event Client_provided_credential(SecretKey, bitstring (* sk proves identity,
       m_presentation *)).
51 event Server_received_credential(SecretKey, bitstring (* sk proves identity,
       m_presentation *)).
52
53 (* Agreement Events *)
54 event Server_has_agreed(SecretKey, bitstring, bitstring (* sk_s, m_R_agreement,
       m_VP_agreement *)).
55 event Client_has_agreed(SecretKey, bitstring, bitstring (* sk_c, m_R_agreement,
       m_VP_requirement *)).
56 event Server_has_offered(SecretKey, bitstring, bitstring, bitstring (* sk_s, m_shape,
       m_R_offer, m_VP_request *)).
57
58 (* Non-Repudiation Events *)
59 event Client_can_prove_server_has_offered(bitstring, bitstring, bitstring, bitstring,
       bitstring, bitstring (* m_shape, sig_shape, m_R_offer, sig_R_offer, m_VP_request,
       sig_VP_request *)).
60 event Client_can_prove_client_has_agreed(bitstring, bitstring, bitstring, bitstring (*
       m_R_agreement, sig_R_agreement_client, m_VP_agreement, sig_VP_agreement *)).
61 event Client_can_prove_server_has_agreed(bitstring, bitstring, bitstring, bitstring (*
       m_R_agreement, sig_R_agreement_server, m_VP_agreement, sig_VP_agreement *)).
```

```
62 event Server_can_prove_server_has_offered(bitstring, bitstring, bitstring, bitstring,
       bitstring, bitstring (* m_shape, sig_shape, m_R_offer, sig_R_offer, m_VP_request,
       sig_VP_request *)).
63 event Server_can_prove_client_has_agreed(bitstring, bitstring, bitstring, bitstring (*
       m_R_agreement, sig_R_agreement_client, m_VP_requirement, sig_VP_requirement
       *)).
64 event Server_can_prove_server_has_agreed(bitstring, bitstring, bitstring, bitstring (*
       m_R_agreement, sig_R_agreement_server, m_VP_agreement, sig_VP_agreement *)).
65
66
67
68 (* START process macros ~ agent *)
69
70 (** START Client **)
71 let client (sk_c:SecretKey, pk_s:PublicKey, talksOnlyToHonest:bool) (* input ~ what the
       agent starts with in private*) =
72
73     new x:exponent;
74     let gx = g(x) in
75     let m'_0 = bitG(gx) in
76     let m_0 = m'_0 in
77     out(h, m_0);
78     (* Sent out Handshake-Request *)
79
80
81     (* Receive Handshake-Response *)
82     in(h,m_1:bitstring);
83     let (eGY:bitstring,m:bitstring) = m_1 in
84     let gy = unbitG(eGY) in
85     let K = GtoSymK(exp(gy,x)) in
86     let s_K = dec(m,K) in
87     if check((gy,gx), s_K, pk_s) then
88     let m'_2 = (sig((gx,gy), sk_c), m_uri) in
89     let m_2 = enc(m'_2, K) in
90     out(h,m_2);
91     (* Sent out data request (enc) *)
92
93
94     (* Receive data usage offer (enc) *)
95     in(h,m_3:bitstring);
96     (* a. decrypt *)
```

```
97  let m'_3 = dec(m_3,K) in
98  (* b. deconstruct *)
99  let (
100     m'_uri:bitstring,
101     sig_shape:bitstring,
102     sig_R_offer:bitstring,
103     sig_VP_request:bitstring
104     ) = m'_3 in
105 (* c. check signatures *)
106 if check(m_shape, sig_shape, pk_s) then
107 if check(m_R_offer, sig_R_offer, pk_s) then
108 if check(m_VP_request, sig_VP_request, pk_s) then
109 (* d. check content *)
110 if m'_uri = m_uri then
111 (* e. create new message content *)
112 let m_presentation:bitstring = m_shape in
113 let m_R_agreement:bitstring = m_R_offer in
114 let m_VP_requirement:bitstring = m_VP_request in
115 (* f. sign files *)
116 let sig_presentation: bitstring = sig(m_presentation, sk_c) in
117 let sig_R_agreement_client:bitstring = sig(m_R_agreement, sk_c) in
118 let sig_VP_requirement:bitstring = sig(m_VP_requirement, sk_c) in
119 (* g. encrypt *)
120 let m_4:bitstring = enc((m_uri, m_presentation, sig_presentation, m_R_agreement,
    sig_R_agreement_client, m_VP_requirement, sig_VP_requirement), K) in
121 (* h. send new message *)
122 event Client_has_agreed(sk_c, m_R_agreement, m_VP_requirement);
123 out(h,m_4);
124 (* Sent out Presentation and Policies *)
125
126
127 (* Client receives Resource and Agreements *)
128 in(h,m_5:bitstring);
129 (* a. decrypt *)
130 let m'_5 = dec(m_5,K) in
131 (* b. deconstruct *)
132 let (
133     m_resource:bitstring,
134     m_VP_agreement:bitstring,
135     sig_VP_agreement:bitstring,
136     m'_R_agreement:bitstring,
```

```
137         sig'_Ragreement_client:bitstring,
138         sig_R_agreement_server:bitstring
139         ) = m'_5 in
140     (* c. check signatures *)
141     if check(m_VP_agreement, sig_VP_agreement, pk_s) then
142     if check(m'_R_agreement, sig_R_agreement_server, pk_s) then
143     (* d. check content *)
144     if m_VP_agreement = m_VP_requirement then
145     if m'_R_agreement = m_R_agreement then
146     if sig'_Ragreement_client = sig_R_agreement_server then
147     (* End of data sharing *)
148
149     (* Client about to complete protocol! *)
150     out(h, Client_completes);
151     if talksOnlyToHonest then
152     out(ch(m_resource),resource_is_private) |
153     (* Events at the end of the protocol *)
154     event Client_provided_credential(sk_c, m_presentation) |
155     event Client_received_resource(sk_c, m_resource) |
156     event Client_can_prove_server_has_offered(m_shape, sig_shape, m_R_offer,
        sig_R_offer, m_VP_request, sig_VP_request) |
157     event Client_can_prove_server_has_agreed(m_R_agreement,
        sig_R_agreement_server, m_VP_agreement, sig_VP_agreement ) |
158     event Client_can_prove_client_has_agreed(m_R_agreement,
        sig_R_agreement_client, m_VP_agreement, sig_VP_agreement);
159     0.
160 (** END Client **)
161
162 (** START Server **)
163 let server(sk_s:SecretKey, pk_c:PublicKey, talksOnlyToHonest:bool) (* input ~ what the
        agent starts with in private*) =
164
165     (* Receive Handshake-Request *)
166     in(h, m_0:bitstring);
167     let gx= unbitG(m_0) in
168     new y:exponent;
169     let gy = g(y) in
170     let K = GtoSymK(exp(gx,y)) in
171     let m'_1 = sig((gy,gx),sk_s) in
172     let m_1 = (bitG(gy),enc(m'_1, K)) in
173     out(h,m_1);
```

```
174     (* Sent out Handshake-Response *)
175
176
177     (* Receive data request *)
178     in(h,m_2:bitstring);
179     (* ab. decrypt deconstruct *)
180     let (
181         sig_K:bitstring,
182         m'_uri:bitstring
183         ) = dec(m_2,K) in
184     (* out-of-band - check if public *)
185     (* c. check signatures *)
186     if check((gx,gy),sig_K,pk_c) then
187     (* d. check content *)
188     if m'_uri = m_uri then
189     (* e. create new message content *)
190     (* f. sign message content *)
191     let sig_shape:bitstring = sig(m_shape, sk_s) in
192     let sig_R_offer:bitstring = sig(m_R_offer, sk_s) in
193     let sig_VP_request:bitstring = sig(m_VP_request, sk_s) in
194     (* g. encrypt *)
195     let m_3:bitstring = enc((m_uri, m_shape, sig_shape, m_R_offer, sig_R_offer,
        m_VP_request, sig_VP_request), K) in
196     (* h. send new message *)
197     event Server_has_offered(sk_s, m_shape, m_R_offer, m_VP_request);
198     out(h, m_3);
199     (* Sent out data usage offer *)
200
201
202     (* Receive Presentation and Policies *)
203     in(h,m_4:bitstring);
204     (* a. decyrpt *)
205     let m'_4 = dec(m_4, K) in
206     (* b. deconstruct *)
207     let (
208         m''_uri:bitstring,
209         m_presentation:bitstring,
210         sig_presentation:bitstring,
211         m_R_agreement:bitstring,
212         sig_R_agreement_client:bitstring,
213         m_VP_requirement:bitstring,
```

```
214         sig_VP_requirement:bitstring
215         ) = m'_4 in
216     (* c. check signatures *)
217     if check(m_presentation, sig_presentation, pk_c) then
218     if check(m_R_agreement, sig_R_agreement_client, pk_c) then
219     if check(m_VP_requirement, sig_VP_requirement, pk_c) then
220     (* d. check content *)
221     if m''_uri = m_uri then
222     if m_R_agreement = m_R_offer then
223     if m_VP_requirement = m_VP_request then
224     if m_presentation = m_shape then
225     (* e. create new message content *)
226     new m_resource:bitstring;
227     let m_VP_agreement:bitstring = m_VP_requirement in
228     (* f. sign message content *)
229     let sig_VP_agreement = sig(m_VP_agreement, sk_s) in
230     let sig_R_agreement_server = sig((m_R_agreement, sig_R_agreement_client), sk_s)
        in
231     (* g. encyrpt *)
232     let m_5:bitstring = enc((m_resource, m_VP_agreement, sig_VP_agreement,
        m_R_agreement, sig_R_agreement_client, sig_R_agreement_server), K) in
233     (* h. send new message *)
234     event Server_has_agreed(sk_s, m_R_agreement, m_VP_agreement) |
235     out (h,m_5);
236     (* Sent out Resource and Rgreements *)
237
238
239     (* Server about to complete the protocol *)
240     out(h, Server_completes);
241     if talksOnlyToHonest then
242     out(ch(m_resource),resource_is_private) |
243     (* Events at the end of the protocol *)
244     event Server_received_credential(sk_s, m_presentation) |
245     event Server_provided_resource(sk_s, m_resource) |
246     event Server_can_prove_server_has_offered(m_shape, sig_shape, m_R_offer,
        sig_R_offer, m_VP_request, sig_VP_request) |
247     event Server_can_prove_client_has_agreed(m_R_agreement,
        sig_R_agreement_client, m_VP_requirement, sig_VP_requirement) |
248     event Server_can_prove_server_has_agreed(m_R_agreement,
        sig_R_agreement_server, m_VP_agreement, sig_VP_agreement);
249     0.
```

```
250 (** END Server **)
251
252 (* END process macros *)
253
254 (* "Do the processes complete?" - sanity check*)
255 query attacker(Client_completes).
256 query attacker(Server_completes).
257 query attacker(resource_is_private).
258
259
260 (* Checking for "Non-repudiation" *)
261
262 query sk_s:SecretKey, policyVP:bitstring, policyR:bitstring, sig_policyR:bitstring,
      sig_policyVP:bitstring;
263     inj-event(Client_can_prove_server_has_agreed(policyR, sig_policyR, policyVP,
      sig_policyVP))
264     ==>
265     inj-event(Server_has_agreed(sk_s, policyR, policyVP))
266     .
267
268 query sk_c:SecretKey, policyVP:bitstring, policyR:bitstring, sig_policyR:bitstring,
      sig_policyVP:bitstring;
269     inj-event(Server_can_prove_client_has_agreed(policyR, sig_policyR, policyVP,
      sig_policyVP))
270     ==>
271     inj-event(Client_has_agreed(sk_c, policyR, policyVP))
272     .
273
274 query sk_s:SecretKey, policyVP:bitstring, policyR:bitstring, sig_policyR:bitstring,
      sig_policyVP:bitstring, credential:bitstring, sig_credential:bitstring;
275     inj-event(Client_can_prove_server_has_offered(credential, sig_credential, policyR,
      sig_policyR, policyVP, sig_policyVP))
276     ==>
277     inj-event(Server_has_offered(sk_s, credential, policyR, policyVP))
278     .
279
280 query sk_s:SecretKey, policyVP:bitstring, policyR:bitstring, sig_policyR:bitstring,
      sig_policyVP:bitstring, credential:bitstring, sig_credential:bitstring;
281     inj-event(Server_can_prove_server_has_offered(credential, sig_credential, policyR,
      sig_policyR, policyVP, sig_policyVP))
282     ==>
```

```
283     inj-event(Server_has_offered(sk_s, credential, policyR, policyVP))
284     .
285
286 query sk_s:SecretKey, policyVP:bitstring, policyR:bitstring, sig_policyR:bitstring,
        sig_policyVP:bitstring;
287     inj-event(Server_can_prove_server_has_agreed(policyR, sig_policyR, policyVP,
        sig_policyVP))
288     ==>
289     inj-event(Server_has_agreed(sk_s, policyR, policyVP))
290     .
291
292 query sk_c:SecretKey, policyVP:bitstring, policyR:bitstring, sig_policyR:bitstring,
        sig_policyVP:bitstring;
293     inj-event(Client_can_prove_client_has_agreed(policyR, sig_policyR, policyVP,
        sig_policyVP))
294     ==>
295     inj-event(Client_has_agreed(sk_c, policyR, policyVP))
296     .
297
298
299 (* IF protocol fullfilled , everyone agreed *)
300 query sk_c:SecretKey, sk_s:SecretKey, credential:bitstring, policyR:bitstring,
        policyVP:bitstring, resource:bitstring;
301     inj-event(Client_received_resource(sk_c, resource))
302     ==>
303         ( inj-event(Server_provided_resource(sk_s, resource))
304         ==>
305             ( inj-event(Server_received_credential(sk_s, credential))
306             ==>
307                 ( inj-event(Client_provided_credential(sk_c, credential))
308                 ==>
309                     ( inj-event(Server_has_agreed(sk_s, policyR, policyVP))
310                     ==>
311                         ( inj-event(Client_has_agreed(sk_c, policyR, policyVP))
312                         ==>
313                         inj-event(Server_has_offered(sk_s, credential, policyR, policyVP))
314                         )
315                     )
316                 )
317             )
318         )
```

```
319     .
320
321
322 (* MAIN *)
323
324 free sk_e:SecretKey.
325 free sk_c:SecretKey [private].
326 free sk_s:SecretKey [private].
327
328 process (*use root process for setup *)
329     (*advertise public keys*)
330     out(h,pk(sk_c));
331     out(h,pk(sk_s));
332     (*let agents dance*)
333     ( (
334           !client     (sk_c, pk(sk_s), true)
335         | !client     (sk_c, pk(sk_e), false)
336         | !server     (sk_s, pk(sk_c), true)
337         | !server     (sk_s, pk(sk_e), false)
338         (* dishonest clients and dishonest servers are impersonated by "the attacker" *)
339     ) )
```

A.2 ProVerif result excerpt - Verifiable Consent Protocol

Listing 30: ProVerif result excerpt from the Verifiable Consent Protocol

```
1  Verification summary:
2
3  Query not attacker(Client_completes[]) is false.
4
5  Query not attacker(Server_completes[]) is false.
6
7  Query not attacker(resource_is_private[]) is true.
8
9  Query inj-event(Client_can_prove_server_has_agreed(policyR, sig_policyR, policyVP,
       sig_policyVP))
10   ==> inj-event(Server_has_agreed(sk_s_3, policyR, policyVP)) is true.
11
12 Query inj-event(Server_can_prove_client_has_agreed(policyR, sig_policyR, policyVP,
       sig_policyVP))
13   ==> inj-event(Client_has_agreed(sk_c_3, policyR, policyVP)) is true.
14
```

```
15 Query inj-event(Client_can_prove_server_has_offered(credential, sig_credential, policyR,
       sig_policyR, policyVP, sig_policyVP))
16   ==> inj-event(Server_has_offered(sk_s_3, credential, policyR, policyVP)) is true.
17
18 Query inj-event(Server_can_prove_server_has_offered(credential, sig_credential, policyR,
       sig_policyR, policyVP, sig_policyVP))
19   ==> inj-event(Server_has_offered(sk_s_3, credential, policyR, policyVP)) is true.
20
21 Query inj-event(Server_can_prove_server_has_agreed(policyR, sig_policyR, policyVP,
       sig_policyVP))
22   ==> inj-event(Server_has_agreed(sk_s_3, policyR, policyVP)) is true.
23
24 Query inj-event(Client_can_prove_client_has_agreed(policyR, sig_policyR, policyVP,
       sig_policyVP))
25   ==> inj-event(Client_has_agreed(sk_c_3, policyR, policyVP)) is true.
26
27 Query inj-event(Client_received_resource(sk_c_3, resource))
28   ==> (inj-event(Server_provided_resource(sk_s_3, resource))
29   ==> (inj-event(Server_received_credential(sk_s_3, credential))
30   ==> (inj-event(Client_provided_credential(sk_c_3, credential))
31   ==> (inj-event(Server_has_agreed(sk_s_3, policyR, policyVP))
32   ==> (inj-event(Client_has_agreed(sk_c_3, policyR, policyVP))
33   ==> inj-event(Server_has_offered(sk_s_3, credential, policyR, policyVP)
34       ))))))  is  true.
```

References

[1] 1, J. T. C. I. J. ISO/IEC 10181-4:1997(en), Information technology — Open Systems Interconnection — Security frameworks for open systems: Non-repudiation framework — Part 4. International Standard, 1997.

[2] Akaichi, I., and Kirrane, S. A Semantic Policy Language for Usage Control. In *Proceedings of Poster and Demo Track and Workshop Track of the 18th International Conference on Semantic Systems co-located with 18th International Conference on Semantic Systems (SEMANTiCS 2022), Vienna, Austria, September 13th to 15th, 2022* (2022), U. Simsek, D. Chaves-Fraga, T. Pellegrini, and S. Vahdat, Eds., vol. 3235 of *CEUR Workshop Proceedings*, CEUR-WS.org.

[3] Baker, T. ShEx · shexSpec/shex Wiki. Tech. rep., Feb. 2019.

[4] Berners-Lee, T., Fielding, R. T., and Masinter, L. M. Uniform Resource Identifier (URI): Generic Syntax. Request for Comments RFC 3986, Internet Engineering Task Force, Jan. 2005. Num Pages: 61.

[5] Berners-Lee, T., Hendler, J., and Lassila, O. The Semantic Web. *Scientific American 284*, 5 (2001), 34–43. Publisher: Scientific American, a division of Nature America, Inc.

[6] Blanchet, B. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends® in Privacy and Security 1*, 1-2 (2016), 1–135.

[7] Blanchet, B., Smyth, B., Cheval, V., and Sylvestre, M. ProVerif 2.04: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial. Software Manual, Nov. 2021.

[8] Bosquet, M. Access Control Policy (ACP). Tech. rep., Sept. 2022.

[9] Braun, C., and Käfer, T. Attribute-based Access Control on Solid Pods using Privacy-friendly Credentials. In *Proceedings of Poster and Demo Track and Workshop Track of the 18th International Conference on Semantic Systems co-located with 18th International Conference on Semantic Systems (SEMANTiCS 2022), Vienna, Austria, September 13th to 15th, 2022* (2022), U. Simsek, D. Chaves-Fraga, T. Pellegrini, and S. Vahdat, Eds., vol. 3235 of *CEUR Workshop Proceedings*, CEUR-WS.org.

[10] Braun, C. H.-J., Papanchev, V., and Käfer, T. SISSI: An Architecture for Semantic Interoperable Self-Sovereign Identity-based Access Control on the Web. In *Proceedings of the ACM Web Conference 2023* (Austin TX USA, Apr. 2023), ACM, pp. 3011–3021.

[11] Brickley, D., and Guha, R. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, Feb. 2004.

[12] Buchner, D., Zundel, B., Riedel, M., and Duffy, K. H. Presentation Exchange 2.X.X. Pre-Draft Specification, Decentralized Identity Foundation (DIF).

[13] Capadisli, S. Web Access Control. Editor's draft, World Wide Web Consortium (W3C), Mar. 2021.

[14] Capadisli, S., Berners-Lee, T., Verborgh, R., and Kjernsmo, K. Solid Protocol. Tech. rep., W3C Community Group, Dec. 2022.

[15] Chadwick, D., Longley, D., Sporny, M., Terbu, O., Zagidulin, D., and Zundel, B. Verifiable Credentials Implementation Guidelines 1.0. W3C Working Group Note, W3C, Sept. 2019.

[16] Esteves, B., Pandit, J. H., and Rodriguez-Doncel, V. ODRL Profile for Access Control. Draft release, W3C, Dec. 2022.

[17] Esteves, B., Rodríguez-Doncel, V., Pandit, H. J., Mondada, N., and McBennett, P. Using the ODRL Profile for Access Control for Solid Pod Resource Governance. In *The Semantic Web: ESWC 2022 Satellite Events* (Cham, 2022), P. Groth, A. Rula, J. Schneider, I. Tiddi, E. Simperl, P. Alexopoulos, R. Hoekstra, M. Alam, A. Dimou, and M. Tamper, Eds., Lecture Notes in Computer Science, Springer International Publishing, pp. 16–20.

[18] European Parliament and the Council, E. P. a. t. C. 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) (Text with EEA relevance), May 2016. Legislative Body: OP_DATPRO.

[19] Fielding, R., Nottingham, M., and Reschke, J. HTTP RFC9110. Tech. rep., June 2022.

[20] Fielding, R., and Reschke, J. F. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Tech. Rep. RFC7231, RFC9110, Internet Engineering Task Force (IETF), June 2014.

[21] Gayo, J. E. L., Prud'hommeaux, E., Boneva, I., and Kontokostas, D. *Validating RDF Data*. Synthesis Lectures on Data, Semantics, and Knowledge. Springer International Publishing, Cham, 2018.

[22] Gillis, A. What is Diffie-Hellman Key Exchange? | TechTarget. *Security* (Oct. 2022).

[23] Hayes, P. J., and Patel-Schneider, P. F. RDF 1.1 Semantics. W3C Recommendation, Apr. 2014.

[24] Iannello, R., and Villata, S. ODRL Information Model 2.2. W3C Recommendation, W3C, Feb. 2018.

[25] Knublauch, H., Inc., T., Kontokostas, D., and of Leipzig, U. Shapes Constraint Language (SHACL). W3C Recommendation, World Wide Web Consortium (W3C), July 2017.

[26] Polleres, A., Esteves, B., and Bos, B. Data Privacy Vocabulary (DPV). Final Community Group Report, Data Privacy Vocabularies and Controls Community Group, May 2022.

[27] Prud'hommeaux, E., Boneva, I., Gayo, J. E. L., Kellogg, G., and Thornton, K. Shape Expressions Language 2.next. Draft Community Report, World Wide Web Consortium (W3C), July 2022.

[28] Shirey, R. W. Internet Security Glossary, Version 2. Request for Comments RFC 4949, Internet Engineering Task Force, Aug. 2007. Num Pages: 365.

[29] Sporny, M., Noble, G., Longley, D., Burnett, D. C., Zundel, B., and Den Hartog, K. Verifiable Credentials Data Model v1.1. W3C Recommendation, World Wide Web Consortium (W3C), Mar. 2022.

Assertion

I declare truthfully that I have written the thesis independently, that I have indicated all aids used completely and correctly, and that I have marked everything that has been taken over unchanged or changed from external work, and that I have observed the KIT Statutes for Ensuring Good Scientific Practice in the currently valid version.

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, August 1, 2023 Valentin Betz