The Rights Delegation Proxy: An Approach for Delegations in the Solid Dataspace

Sebastian Schmid[1], Daniel Schraudner[1] and Andreas Harth[1,2]

[1]*Friedrich-Alexander-Universität Erlangen-Nürnberg, Nuremberg, Germany*
[2]*Fraunhofer Institute for Integrated Circuits IIS, Nuremberg, Germany*

Abstract
We propose the Rights Delegation Proxy to check and execute delegations in dataspaces building on the Social Linked Data project (Solid). The Rights Delegation Proxy ensures privacy by keeping delegation details hidden and validates delegated actions against policies for legitimacy. We show our implemented architecture in a loan contract scenario, where a person signs a contract on behalf of a company with a bank. Additionally, we analyze the flow of our architecture for privacy and legitimacy using formal models.

1. Introduction

Agents act in their environments to reach defined goals, often their own [1]. Agents, however, may also have to act on behalf of others, e.g. natural persons on behalf of organizations (e.g. as representative or procurator), employees on behalf of the superiors (e.g. task to fulfill), and colleagues on behalf of their fellows (e.g. as a substitute). We call the transfer of rights and responsibilities from one party to another a delegation or power of attorney [2, 3].

As Solid [4] started to bring identifiable agents to the Web, the concept of data sharing and the creation of organizations and communications are starting to be well defined. But while the idea of Solid dataspaces (SDS) [5] gains momentum, the integral aspect of delegation among agents is still open. For organizations with hierarchies and sub-organizations to thrive in dataspaces, organizations need actions and rights to be delegated along complex structures. Addressing the gap of delegation, we ask: How can delegation among agents find its way into the Solid dataspace? Delegations refer to agents, e.g. natural persons or legal entities like companies, and have complex specifications for defined cases, e.g. in business relations for signing on behalf of a company up to a defined sum of money. We define the following roles and parts of a delegation:

- **Affiliate:** an agent that receives transactions
- **Policy:** defined rights that may be exercised in transactions towards an affiliate
- **Delegate:** an agent that acts based on a policy towards an affiliate

The Second International Workshop on Semantics in Dataspaces, co-located with the Extended Semantic Web Conference, May 26 – 27, 2024, Hersonissos, Greece
sebastian.schmid@fau.de (S. Schmid); daniel.schraudner@fau.de (D. Schraudner); andreas.harth@iis.fraunhofer.de (A. Harth)
0000-0002-5836-3029 (S. Schmid); 0000-0002-2660-676X (D. Schraudner); 0000-0002-0702-510X (A. Harth)

- **Delegator:** an agent that defines a policy for a delegate to act in the delegator's name

An example of a delegation is a company SME (delegator) that grants its employee Alice (delegate) the right to sign contracts on its behalf (policy) with BigBank (affiliate). Considering the requirements delegators have for a delegation, we obtain three basic points:

R1 **Privacy**: An affiliate shall not necessarily be informed, whether a delegation has happened and who is the delegate.

R2 **Legitimacy**: Every taken action by the delegate shall be validated against the policies as defined by the delegator and only approved actions may happen.

R3 **Completeness**: Every initiated action by a potential delegate shall receive a response.

A conventional delegation process in SDS may use access and manipulation rights for resources based on Web Access Control (WAC)[1], e.g. read, write, append rights based on Access Control Lists (ACL). While ACLs ensure that at least authenticated or specified agents may use the granted rights, only small extensions for more complex agent structures are possible by using vCard groups[2] in Solid. With groups, the hierarchy has to stay flat as no nesting of groups and rights is possible, while the members of a group need to be shared with external affiliates who need to check if acting agents are part of a group defined in the ACL.

The conventional delegation process is even less attractive when considering privacy issues (cf. General Data Protection Regulation [6]) - there is neither a per se necessity for a delegator to share that an action was delegated nor the need to share the delegate's identity. A delegate's identity and the delegation act are revealed to an affiliate who has to set the corresponding ACLs, despite, the delegate's potential interest to stay secret towards an affiliate. The affiliate shall only know that the action was taken and to that extent that the party that was originally granted that right, the delegator, has taken the action.

Dataspace proposals like the International Data Space (IDS) [7] or GAIA-X [8] try to solve delegations via connectors, however, introduce complexities by building on specific architectures and custom protocols instead of using established Web technologies, but promise to offer trustworthy and sovereign data sharing instead [5]. An extension of IDS and GAIA-X with verifiable credentials can address delegations of data processing [9], but the question is left unanswered how much privacy remains for delegators and delegates in an IDS delegation. Solid in its current state implements a solution based on ACLs on a resource- and WebID-bound level, which is insufficient for delegations. The possibility to declare rights for groups of WebIDs lacks the sophistication to mirror more complex hierarchies. Besides, the SDS offers a Web-based solution for dataspaces instead of custom protocols. All in all, current solutions lack an elaborated mechanism to ensure a GDPR-conformant solution that separates the identities of delegator and delegate, as well as the possibility of keeping the delegation itself hidden.

As we see the SDS approach as the most promising to realize dataspaces, because of the SDS foundation on Web standards, easy adoption, interoperability, and inherently decentralized architecture, we propose to extend Solid's architecture on the dataspace application layer, where currently aspects of certification and policy enforcement are lacking [5]: we present the Rights Delegation Proxy (RDP) as an approach for private and legitimate data sharing and delegations.

[1] https://solidproject.org/TR/wac

[2] https://www.w3.org/2006/vcard/ns#Group

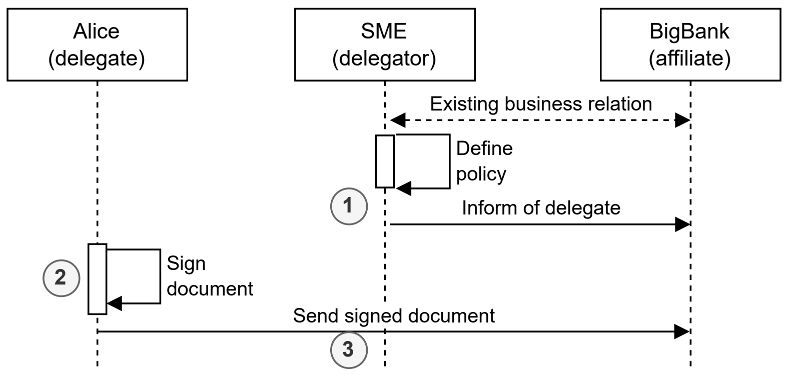

Figure 1: Conventional delegation process for a loan scenario (cf. Ex. 2): Alice (the *delegate*) signs the loan offer on behalf of SME (the *delegator*) with BigBank (the *affiliate*).

Furthermore, the overall architecture shall be compatible with the inherently decentralized style of the Web and allow automated validation and execution of actions in a delegation. Our contributions are:

- We present the architecture of the Rights Delegation Proxy and its implementation to extend Solid dataspaces that enable privacy and legitimacy for delegations among agents
- We give formal proof to show that our architecture fulfills the requirements of privacy and legitimacy for delegations among agents and compare our architecture formally to a conventional delegation process based on the example of a loan signature delegation

2. Running example

We explain our running example, where two legal entities, the banking institution BigBank and the enterprise SME, and the natural person Alice interact. We show a simplified, conventional delegation process for a loan contract that contains a delegation (see also Fig. 1):

Example 1 Because of a previous business relation, SME received a loan offer from BigBank. Today, the offer has to be signed such that SME gets the loan. SME defines a policy that its employee Alice has the right to sign loan offers from BigBank and informs BigBank about the policy (1). Alice concludes that the offer is beneficial for SME and gives her signature as "on behalf of SME, Alice" (2). The contract is sent back to BigBank (3) and some days later, SME receives the details to access their money.

SME and BigBank are legal entities that cannot act on their own - a natural person has to act "on behalf of" them via a delegation. Here, the natural person Alice acts on behalf of SME where Alice is accepting the offer by giving her signature (that is acting) for SME. Similarly, but not shown here, BigBank has to delegate its actions, e.g. offering the loan contract. Consider the implications that the example loan process hides in terms of delegation:

- SME (the delegator) defines a right to sign loan offers (a policy) for Alice (the delegate) via her role in the organization and thus creates a delegation.

- The delegation to Alice is an SME internal step that is shared externally, here with BigBank (an affiliate). The loss of privacy is necessary to declare that Alice may sign on behalf of SME, otherwise BigBank would receive a signature from an unknown natural person called Alice who only claims to act on behalf of SME.
- Alice could have signed even if she did not receive a delegation. Although the misconduct could be detected later on, loss of reputation and money can be the consequence.

3. Background and related work

3.1. The Solid project

Solid is a collection of technological specifications for read/write Linked Data supported by authentication and authorization [4]. Solid builds on the RESTful HTTP specification of Linked Data Platform (LDP)[3] for manipulating and controlling data resources, and on (Semantic) Web standards [10]. Solid aims to be easy to integrate by using lightweight specifications without restrictions to the used database backend. Access control to resources is realized via WAC or ACP. We briefly define two crucial concepts of Solid:

- **Solid storage:**[4] a place for storing resources managed by access control. The physical location and identity provider can be freely chosen or self-hosted.
- **Solid agent:**[5] a person, social entity, or software identified by an HTTP URI called WebID[6], including a profile to uniquely describe the denoted agent.

3.2. Dataspaces and Solid dataspaces

The idea of dataspaces shall help to manage multiple data sources with multiple schemas [11], mostly for single stakeholders that lack control or ownership across multiple required data sources. Initiatives like the International Data Space (IDS) [7] or GAIA-X [8] picked up dataspaces for datasharing that uses standards, governance models, and a secure and standardized data exchange, with a focus on multiple participants across organizations and data sovereignty [12]. Meckler et al. [5] identified Solid as a viable framework for decentralized dataspaces, called Solid dataspaces: Solid builds on established technologies for interoperability like the Web for communication and Linked Data for data, while Solid itself specifies the interaction, and dataspaces on top of Solid can be extended for additional processes and protocols.

3.3. Delegation and power of attorney

Concerning the power of attorney, an agent (called delegate or attorney-in-fact) may be granted the authority to act on behalf of someone else (called delegator or principal) [2], e.g. usually an organization or entity, and make decisions or enter into agreements on their behalf. While national laws of delegation and the range may vary, the authority for business affairs includes

[3]https://www.w3.org/TR/ldp/
[4]formerly known as Solid Personal Online Data (Pod) (https://solidproject.org/TR/protocol#storage)
[5]https://solidproject.org/TR/protocol#agent
[6]https://www.w3.org/2005/Incubator/webid/spec/tls/

e.g. contract negotiation and signature rights [3]. The limits of a power of attorney have to be considered and stated carefully, as abuse of power or lost income may be the consequence [2].

4. Approach

4.1. Solid agents

We assume that all involved roles are represented as authenticated Solid agents (see Sec. 3.1). We do not make assumptions on how delegator and delegate share their delegation, or how affiliate and delegator agree on granted rights for the prepared location and regard this communication as domain dependent and thus out of scope.

Affiliate The affiliate stores web resources on a Solid storage that the delegator may access or manipulate. Thus, the affiliate needs to know the delegator's WebID to define ACLs for the Web resource with respective rights for the delegator.

Delegator The delegator defines policies that connect the delegate's WebID and the affiliate's web resource, so the delegator knows the delegate's WebID as well as which applicable rights shall be granted to the delegate. The granted rights may include all or a subset of the delegator's rights towards the affiliate. Additionally, the delegator uses and authorizes an RDP, e.g. as an own service or hosted by a provider, and shares where policies and logs are stored.

Delegate The delegate ideally (but not necessarily) knows the delegator's granted rights, the URI of the affiliate's web resource that shall be accessed or manipulated, and the RDP's URI such that the delegation may be enacted.

4.2. Running example revisited

Compared to the conventional delegation process of Sec. 2, our architecture uses the Solid dataspace where an Rights Delegation Proxy acts as a mediator between the legal entities and natural persons. Again, Alice shall sign a loan contract with BigBank for SME (see also Fig. 2):

Example 2 SME has to sign a loan offer from BigBank to get the loan. To receive the signature, BigBank prepared a resource `https://bankpod.net/signHere` that shall hold the digital signature. As BigBank expects SME to sign the contract, BigBank created an ACL `signHere.acl` with `acl:Read` and `acl:Write` rights exclusively for SME's WebID `https://smepod.net/profile/card#me`.

SME defines a policy at `https://smepod.net/policies` that Alice, is authenticated by her WebID `https://alice.solidcommunity.net`, has the right to sign loan offers from BigBank (1). Alice sends an HTTP PUT request *to the RDP* pointing to `https://bankpod.net/signHere` to give the signature (2). The RDP receives Alice's authenticated request and her WebID. RDP gets (3) and checks (4) the SME's defined policies if Alice's WebID has a granted delegation for `https://bankpod.net/signHere`, and logs the received request (5).

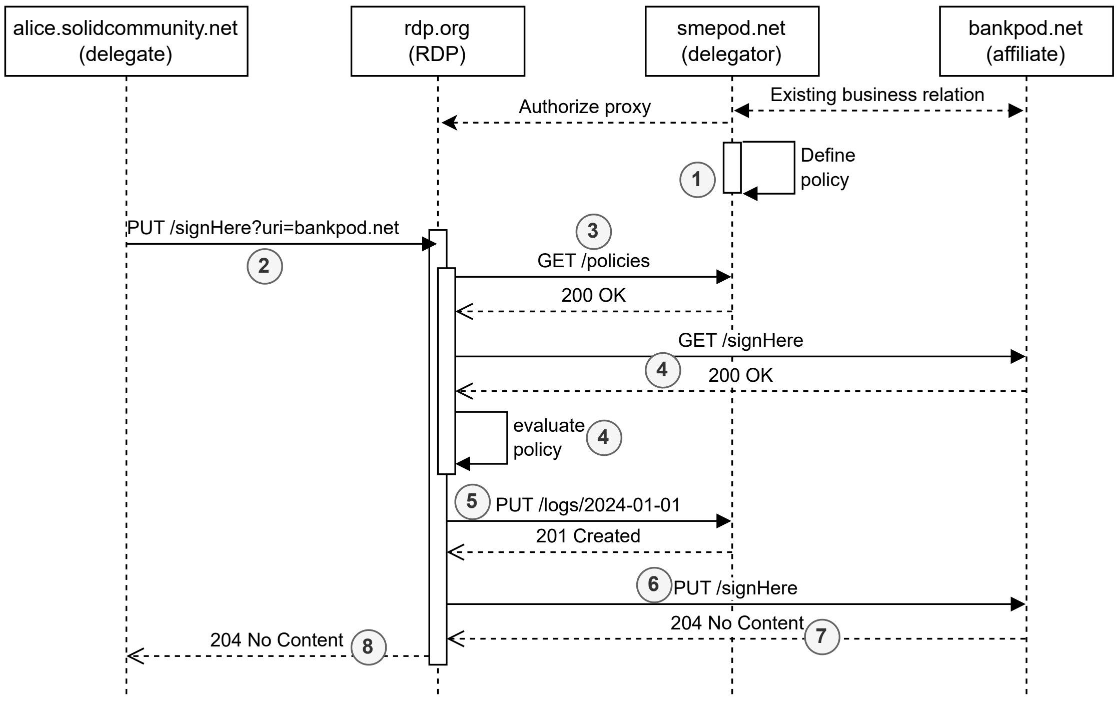

Figure 2: The Rights Delegation Proxy receives a request and checks the policies (steps 1-4). After checking the conditions, the RDP logs and forwards Alice's request, and returns the response (steps 5-8)

As Alice fulfills the policy, RDP forwards Alice's request to the specified `https://bankpod.net/signHere` (6), but changes the authentication to SME. At BigBank, an authenticated request arrives from SME's WebID. As the ACL is set for SME's WebID the signature is created to accept the loan (7), and the response forwarded to Alice (8).

From BigBank's perspective, only SME was involved as the request came from SME's WebID. But while the legal body of SME could still not act on its own, Alice initiated the action which resulted in an accepted offer. Again, we consider the implications:

- SME delegates to sign the contract with BigBank to Alice via a defined policy. The delegation is not shared with BigBank.
- BigBank knows and anticipates only SME, so the ACL lets only SME interact with the protected resource, but not Alice.
- The signature comes from SME directly after being checked against the policies. SME has thus been informed and approved of the interaction.
- Alice may only sign after SME defines and approves the delegation via a policy. Without a valid policy, the RDP does not forward Alice's request. A request from Alice directly to BigBank would be blocked as unauthorized from the ACL.

4.3. Implementation of the Rights Delegation Proxy architecture

4.3.1. Rights Delegation Proxy

Below, we present an overview of the interactions between the Solid agents and the RDP during the delegation process. We use Fig. 2 to illustrate how the delegation works for our running example from Sec. 4.2.

1. The delegator defines a policy that states the delegate's rights of transactions towards the affiliate. We discern two types of policies that are evaluated: pre-conditions and post-conditions.

- **Pre-conditions** define how a to-be-accessed resource has to look like *before* the delegate may access it.
- **Post-conditions** define how a resource has to look like *after* the delegate accessed it.

Fig. 2, step 1: The delegator SME creates a policy at `https://smepod.net/policies` for `https://alice.solidcommunity.net/profile/card#me`. The policy contains a pre- and post-condition as Shape Expressions (ShEx)[7] that allow the structural description and validation of data, where an example policy could look like as follows:

```
@prefix ex: <https://example.org/vocab#> .
<> a ex:Policy ;
  ex:delegator <https://smepod.net/profile/card#me> ;
  ex:delegate <https://alice.solidcommunity.net/profile/card#me> ;
  ex:preCondition <LoanContractShape> ;
  ex:postCondition <LoanContractSignedShape> .
```

The ShEx shapes `LoanContractShape` and `LoanContractSignedShape` are given as:

LoanContractShape	LoanContractSignedShape

<LoanContractShape> CLOSED{
ex:forCustomer [sme:me];
a [ex:LoanContract] }
	<LoanContractSignedShape> CLOSED{
	ex:forCustomer [sme:me];
	a [ex:LoanContract];
	ex:signed [true] }

The affiliate created a resource at `https://bankpod.net/signHere` together with an ACL with read and write rights for SME:

https://bankpod.net/signHere	https://bankpod.net/signHere.acl
<> a ex:LoanContract ; ex:forCustomer <https://smepod.net/profile/card#me> .	@prefix acl: <https://www.w3.org/ns/auth/acl#> . <> a acl:Authorization ; acl:accessTo <./signHere> ; acl:agent <https://smepod.net/profile/card#me> ; acl:mode acl:Read, acl:Write .

[7] https://shex.io/

2. The delegate makes an authenticated HTTP request to the Rights Delegation Proxy, where the accessed path is equal to the web resource at the affiliate and the query contains the host of the affiliate's URI.

Fig. 2, step 2: Alice is authenticated as `https://alice.solidcommunity.net/profile/card#me` and sends a PUT request to the RDP at `https://rdp.org/signHere?uri=https://bankpod.net` as Alice wants to access `https://bankpod.net/signHere` as a delegate. To sign the contract, the message body contains:

```
<> a ex:LoanContract ;
ex:forCustomer <https://smepod.net/profile/card#me> ;
ex:signed true .
```

3. The RDP receives the request, verifies the delegate's identity using the Solid-OIDC WebID claim, and extracts the affiliate's URI as well as the path to the web resource. The RDP looks up suiting policies at the delegator depending on the WebID or web resource.

Fig. 2, step 3: The RDP receives the HTTP PUT request from `https://alice.solidcommunity.net/profile/card#me`, verifies the identity, and retrieves defined policies from `https://smepod.net/policies` for Alice's WebID.

4. The RDP evaluates if the delegate's request is valid concerning the policy's conditions.

- To check a pre-condition, the RDP does a "preflight GET request" to the requested resource and evaluates if the response matches the pre-condition. If not, the RDP responds to the delegate with a *403 Forbidden*.
- To check a post-condition, the RDP evaluates if the message body, which contains the to-be-expected resource state, adheres to the post-condition. If the post-condition does not hold, the RDP responds with *403 Forbidden*.

Fig. 2, step 4: In the retrieved policy, RDP finds two ShEx shapes as pre- and post-condition. For the pre-condition, RDP does a "preflight" GET request to `https://bankpod.net/signHere`, validates the retrieved state with LoanContractShape, and finds it valid. For the post-condition, RDP extracts the content of the PUT request, validates it with LoanContractSignedShape, and finds it valid, too.

5. After checking the request, the RDP logs the applied policy, its result as well as time, content, accessed resource, and requesting WebID at a location defined by the delegator.

Fig. 2, step 5: The RDP logs Alice's request to `https://bankpod.net/signHere` at `https://smepod.net/logs/2024-01-01` with:

```
<> a ex:Log ;
ex:checkedPolicy (<LoanContractShape> <LoanContractSignedShape>);
ex:forAgent <https://alice.solidcommunity.net/profile/card#me> ;
ex:forRessource <https://bankpod.net/signHere> .
```

6. The RDP authenticates as delegator to forward the checked request to the resource as specified by the delegate in the query string.

Fig. 2, step 6: The RDP authenticates as `https://smepod.net/profile/card#me` via Solid OIDC and forwards Alice's HTTP PUT request to `https://bankpod.net/signHere` such that the state is now:

```
<> a ex:LoanContract ;
ex:forCustomer <https://smepod.net/profile/card#me> ;
ex:signed true .
```

7. The affiliate responds to the RDP as the sender of the forwarded request. The RDP logs the response.

Fig. 2, step 7: The affiliate `https://bankpod.net` responds to the RDP `https://rdp.org` with *204 No Content* as the HTTP PUT was enacted as requested.

8. The RDP sends the affiliate's response to the delegate and concludes the flow of messages for the initiated request.

Fig. 2, step 8: The RDP `https://rdp.org` sends the affiliate's response *204 No Content* back to Alice.

As a remark, if the affiliate responds with error codes, e.g. *500 Internal Server Error*, the RDP forwards the messages to the client and logs the error.

4.3.2. Implementation

Our implementation for the RDP is available online [8]. We use *node.js*[9] and *express.js*[10] as a web framework for the RDP server side and use the OIDC library *Jose* [11] to handle the Solid authentication. We use *ShEx*[12] to express pre- and post-conditions of policies.

5. Proof

We used ProVerif [13], an established software for protocol verification with formal models, to analyze the RDP approach and present a formal proof for the correctness of our approach. Note that our proof focuses on the correctness of the delegation process via events and makes *no* claims about cryptographic security. We make the explicit assumption that Transport Layer Security and Solid OpenID Connect[13] are correct and functionable during the RDP process. Our ProVerif models, written in applied π-calculus, are available online [14]. ProVerif defines concurrent processes that use channels to communicate with each other. Channels may transport names that act as constants.

[8]https://github.com/wintechis/delegation-proxy
[9]https://nodejs.org/en
[10]https://expressjs.com/
[11]https://github.com/cisco/node-jose
[12]https://shex.io/
[13]https://solidproject.org/TR/oidc
[14]https://github.com/wintechis/delegation-proxy

5.1. Components

We assume the following processes that represent the roles and agents in a delegation: clients C, the RDP R, the delegator D, and the affiliate A.

We define a client as a process that wants to send an operation to the affiliate. In case a client received a delegation before, it is also considered a delegate: we instantiate two clients with a to-be-performed operation *op*, the client C_V (that is also a delegate) and the client C_I (that is *not* a delegate), where C_V is initialized with a name $validName$ and C_I is initialized with $invalidName$. D regards C_V as delegate and knows the client's $validName$ together with an allowed operation *op* which defines the policy. A knows the delegator's $orgName$, also with an allowed operation *op*. R only knows the delegator's name.

We give the system with the RDP in the style of ProVerif[15], where for two processes P and Q we call $P|Q$ a parallel composition of P and Q (processes run in parallel), and $!P$ an infinite composition $P|P| \dots$, which denotes an unbounded number of processes:

$$!(C_V(invalidName, op)|C_I(validName, op)|R(orgName)|D(validName, op)|A(orgName, op))$$

5.2. Checked properties

In Sec. 1 we identified three requirements for delegations from which we derive goals to prove in ProVerif:

R1 **Privacy**: affiliates shall not know whether a delegation has happened
R2 **Legitimacy**: all delegate actions are validated against delegator's policies
R3 **Completeness**: clients shall receive a response to all of their requests

The following goals shall ensure that the delegation process is correct. Each goal G1-3 maps directly to the corresponding requirement R1-3:

G1 The affiliate A will never get to know a client C's name, regardless of whether C's name was a delegate or not according to the delegator D.
G2 For all messages a client C_V or C_I sends to the affiliate A (possibly using the RDP R), the delegator D's policies have been validated.
G3 For all messages a client C_V or C_I sends to the affiliate A (possibly using the RDP R), the client will receive a response afterwards.

Thus, we define events that may occur during the exchange:

- $clientSendName$ - event emitted by a client C after sending a message
- *clientReceiveResponse* - event emitted by a client C after receiving a response
- $orgOK$ - event emitted by the delegator D after providing policies for validation
- *affReceivesMsg* - event emitted by affiliate A after accepting a message

We represent our goals as injective correspondences in ProVerif where $\alpha \rightarrow \beta$ is true for two events α and β if and only if when there is an occurrence of α, then there is a distinct earlier occurrence of β *before* that is α and β have a one-to-one-relationship. For a fact γ, e.g. a name, we check with `attacker(`γ`)` if γ is accessible on a public channel. Here, we check for facts arriving at the affiliate (see Sec. 5.3). We want the following goals to hold as **true**.

[15] https://bblanche.gitlabpages.inria.fr/proverif/manual.pdf

G1 $\neg$`attacker`$(validName)$ `and` $\neg$`attacker`$(invalidName)$
G2 $affReceivesMsg \rightarrow orgOK$
G3 $clientReceiveResponse \rightarrow clientSendName$

5.3. Modeled processes for RDP architecture

We give the processes of Sec. 5.1 in detail, as used in ProVerif. With Solid OIDC, we assume the channels between client↔RDP, RDP ↔ delegator, and RDP↔affiliate as private. Sending a name via a channel is equal to being authenticated via the name's respective identity provider.

Note that we create a *nonce* at the client at the beginning of the process, similar to a session ID. While there is no explicit nonce in the RDP, we use it in our model to check for the injective correspondence of events. It serves no further purpose.

5.3.1. Client C

The client sends its name and used operator via channel *cout* to the RDP and expects an answer at channel *cin*.

```
let client(clientname:Name, op:bitstring) =
  new nonce:bitstring;
  let m0 = (clientname, op, nonce) in event clientSendName(nonce) ;
  out(cout, m0);
  in(cin, m4:bitstring) ;
  let (resp:bitstring,nonce2:bitstring) = m4 in if nonce = nonce2 then
  event clientReceiveResponse(nonce)  .
```

5.3.2. Rights Delegation Proxy R

The RDP receives a name and desired operator of the clients on channel *cout*. The RDP sends the client's name to the delegator via channel *orgin* to receive a policy as the answer on channel *orgout*. If the policy matches the client name and operator, the delegator's name and operator are sent to the affiliate on channel *affin* and the response received on channel *affout*. The client receives the response to its request on channel *cin*.

```
let rdp(orgName:Name) =
  in(cout, m0:bitstring) ;
  let (xname:Name , xop:bitstring, nonce:bitstring) = m0 in
 let m1 = (xname, nonce) in out(orgin, m1) ; (* send Name to delegator *)
  in(orgout, m2:bitstring) ;
  let (polName:Name, polOp:bitstring, nonce2:bitstring) = m2 in
  if (xname, xop, nonce) = (polName, polOp, nonce2) then
  let m3 = (orgName, xop, nonce) in out(affin, m3 );
  in(affout, m4:bitstring); (* proxy response*)
  let (resp:bitstring,nonce3:bitstring) = m4 in
  if nonce = nonce3 then event rdpWasSuccessful(nonce);
  out(cin, m4) .
```

5.3.3. Delegator D

The delegator receives requests from the RDP via channel $orgin$. The RDP receives a response via channel $orgout$, depending on the relayed name being equal to the initialized value of the delegator that is if it matches a policy.

```
let delegator(okName:Name, okOp:bitstring) =
  in(orgin, m1:bitstring ) ;
  let (yname:Name, nonce:bitstring) = m1 in
  if yname = okName then event orgOK(nonce) ;
  let m2 = (okName, okOp, nonce) in out(orgout, m2) .
```

5.3.4. Affiliate A

The affiliate receives a name and operation from the RDP via the channel *affin* and checks if the name is equal to the known delegator name as well as the operation being allowed. The affiliate responds via channel *affout* and sends the received name to a public channel pub. We use the public channel functionality of ProVerif to check which name reaches the affiliate and which does not (see Sec. 5.2). Note that a client C sending a request directly to A will always result in a negative response, as A only accepts requests carrying the delegator's name.

```
let affiliate(orgName:Name, op:bitstring)  =
  in(affin, m3:bitstring) ;
  let (zname:Name, zop:bitstring, nonce:bitstring) = m3 in
  if (zname, zop) = (orgName, op) then event affReceivesMsg(nonce) ;
  let m4 = (OK,nonce) in out(affout,m4) ;
  out(pub,zname) . (* check if affiliate received client Name*)
```

5.4. Proof results

5.4.1. Results for RDP

ProVerif proved that for the presented architecture of Sec. 5.3 that indeed

G1 $\neg\texttt{attacker}(validName) = \textbf{true}$ `and` $\neg\texttt{attacker}(invalidName) = \textbf{true}$

G2 $\mathit{affReceivesMsg} \rightarrow orgOK = \textbf{true}$

G3 $\mathit{clientReceiveResponse} \rightarrow clientSendName = \textbf{true}$

meaning that the affiliate will not get to know a client's or delegate's name, that a delegate's message using the RDP will always reviewed by the delegator, and that any client will receive a response to a request. These proofs are independent of a client being a delegate.

5.4.2. Comparison to conventional delegation

For comparison, we analyzed a conventional delegation without an RDP, based on Fig. 1, with

$$!(C_V(invalidName, op)|C_I(validName, op)|D(validName, op)|A(orgName, op))$$

where D sends its policy with the delegate's $validName$ to A (similar to setting ACLs to include the delegate), and clients C send their name and operation directly to A. We find that in the conventional delegation

G1 $\neg\texttt{attacker}(validName) = \textbf{false}$ and $\neg\texttt{attacker}(invalidName) = \textbf{false}$
G2 $affReceivesMsg \rightarrow orgOK = \textbf{true}$
G3 $clientReceiveResponse \rightarrow clientSendName = \textbf{true}$

meaning that a client's or delegate's name does **not** stay hidden from the affiliate after receiving a message, that a delegate's message is only accepted after the delegator informs the affiliate of the applied policy, and that any client will receive a response to a request.

6. Discussion

We proposed the RDP to address the gap of more complex delegations in Solid dataspaces. Up to now, only basic possibilities exist to define delegated actions, but with the RDP, we extend the architecture by introducing an active component that checks the actions of delegates while keeping their privacy and still makes sure that every tracked, forwarded action on the delegator's behalf is legitimate.

The RDP acts in its presented state as a necessary medium between the explicitly separated agents delegator, delegate, and affiliate. As all actions of the delegate have to go through the RDP and the RDP can authenticate (like a Solid App) as the delegator, the RDP is a component with high responsibility and no delegated action may happen without it. As a consequence, we shift the power over actions to the delegator by having exclusive control over policies, while shifting responsibility away from the affiliate, who has to know only the delegator and nobody else. The delegate powers are (rightfully) limited to exist only in the defined policies. The enhanced responsibility of RDP thus solves the privacy problem straightforwardly: with authentication as delegator (comparable to the usage of a Solid App through the delegator), there is no differentiation of the action's origins towards a third party, while the delegate's identity is obfuscated. Solid OIDC establishes an authenticated and trusted connection between agents in our architecture.

Admittedly, the RDP as presented in our example appears a centralized component that manages all incoming delegated requests, which presents a bottleneck and a potential single point of failure [14]. Handling a single request, however, is independent of other requests such that multiple instances of an RDP may be run in parallel. Here, a load balancer may distribute incoming requests to several RDP instances to scale for big organizations. The RDP authenticates as the delegator to execute actions which currently needs the delegator to share credentials with the proxy, which might pose a security issue if the RDP is implemented and handled carelessly. For the authentication of the RDP, OIDC offers the possibility to use client credentials, Solid IDPs, however, are not forced to offer client credentials by Solid OIDC.

For illustration, our example in Sec. 4.3.1 uses a simplistic, custom approach with ShEx shapes for policies. The specific implementation is of course subject to the applied use case of the RDP, but we see huge potential for other approaches that make complex, custom policies for large organizations possible: the Open Rights Digital Language[16] may be used to define post-condition via obligations and pre-conditions via constraints. SPARQL[17] ASK queries can be used instead of shapes to evaluate the pre- and post-conditions. If the specified conditions in the query are met (as with shapes), the delegate request is forwarded. As organizations often use Business Process Model and Notation (BPMN) to describe more complex workflows (e.g. a contract may only be signed after an accountant agrees), ontologies like WiLD [15] can be used to represent and monitor workflows similarly and be validated by the RDP. We connect delegates and policies explicitly via WebIDs. For organizations, the connection can be broadened to include memberships in departments or specific roles e.g. modeled in the organization ontology ORG[18]. In any case, the complexity and number of policies have to be chosen carefully as the RDP needs time to validate all policies associated with a request, which may lead to delays.

All in all, we argue that our presented approach to extend the Solid dataspace with an Rights Delegation Proxy component leads to a more flexible, privacy-respecting, secure delegation mechanism that builds directly on existing Web standards.

7. Conclusion and outlook

We present a solution for the gap of possible delegation between agents in the Solid dataspace which we call the Rights Delegation Proxy (RDP). With the Rights Delegation Proxy, a delegating party can define and check policies over actions while a delegate can exercise rights with their privacy respected. An affiliate that receives these actions can still ensure that the actions are taken by the delegating party, as opposed to conventional delegation processes. We base our approach on Solid, Linked Data, and Web technologies to describe and implement an RDP that checks, logs, and forwards HTTP requests of delegates to an affiliate for defined policies by the delegator. We show the system in the context of our running example, a loan contract scenario where a natural person signs a contract on behalf of a company.

In the future, we want to extend the RDP towards more sophisticated policies to define delegations, which includes the possible use of SPARQL ASK queries and reasoning to distinguish internal and external relationships in organizations.

Acknowledgments

This work is partially funded by the German Federal Ministry of Education and Research via the MANDAT project (FKZ 16DTM107A). The authors thank Christoph H.-J. Braun for his advice regarding ProVerif and the models.

[16] https://www.w3.org/TR/odrl-model/
[17] https://www.w3.org/TR/sparql11-overview/
[18] https://www.w3.org/TR/vocab-org/

References

[1] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed., Prentice Hall Press, USA, 2009.

[2] M. M. Hughes, Remedying financial abuse by agents under a power of attorney for finances, Marquette Elder's Advisor 2 (2012) 39.

[3] W. C. Schmidt, Supported decision-making proxy decision-making : A legal perspective, 2015. URL: https://api.semanticscholar.org/CorpusID:53399327.

[4] S. Capadisli, T. Berners-Lee, R. Verborgh, K. Kjernsmo, Solid Protocol, 2021. URL: https://solidproject.org/TR/protocol.

[5] S. Meckler, R. Dorsch, D. Henselmann, A. Harth, The Web and Linked Data as a Solid Foundation for Dataspaces, in: Companion Proceedings of the ACM Web Conference 2023, WWW '23 Companion, Association for Computing Machinery, New York, NY, USA, 2023, p. 1440–1446. URL: https://doi.org/10.1145/3543873.3587616.

[6] Parliament and Council of the European Union, Regulation (EU) 2016/679 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), 2016.

[7] S. Steinbuss, et al., IDS Reference Architecture Model 4. Technical Report, 2024. URL: https://github.com/International-Data-Spaces-Association/IDS-RAM_4_0?tab=readme-ov-file.

[8] Gaia-X , Gaia-X Architecture Document - 22.04 Release, 2022. URL: https://docs.gaia-x.eu/technical-committee/architecture-document/22.04/.

[9] H. Meyer zum Felde, M. Kollenstart, T. Bellebaum, S. Dalmolen, G. Brost, Extending actor models in data spaces, in: Companion Proceedings of the ACM Web Conference 2023, WWW '23 Companion, Association for Computing Machinery, New York, NY, USA, 2023, p. 1447–1451. URL: https://doi.org/10.1145/3543873.3587645.

[10] E. Mansour, A. V. Sambra, S. Hawke, M. Zereba, S. Capadisli, A. Ghanem, A. Aboulnaga, T. Berners-Lee, A Demonstration of the Solid Platform for Social Web Applications, in: Proceedings of the 25th International Conference Companion on World Wide Web - WWW '16 Companion, ACM Press, 2016, pp. 223–226. doi:10.1145/2872518.2890529.

[11] M. Franklin, A. Halevy, D. Maier, From databases to dataspaces: a new abstraction for information management, SIGMOD Rec. 34 (2005) 27–33. URL: https://doi.org/10.1145/1107499.1107502. doi:10.1145/1107499.1107502.

[12] A. Reiberg, C. Niebel, P. Kraemer, What is a Data Space? Definition of the concept Data Space, 2022. URL: https://gaia-x-hub.de/wp-content/uploads/2022/10/White_Paper_Definition_Dataspace_EN.pdf.

[13] B. Blanchet, Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif, Foundations and Trends® in Privacy and Security 1 (2016) 1 – 135. URL: https://inria.hal.science/hal-01423760. doi:10.1561/3300000004.

[14] R. de Lemos, et al., Software Engineering for Self-Adaptive Systems: A Second Research Roadmap, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 1–32.

[15] T. Käfer, A. Harth, Specifying, monitoring, and executing workflows in linked data environments, in: D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, E. Simperl (Eds.), The Semantic Web – ISWC 2018, Springer International Publishing, Cham, 2018, pp. 424–440.