

# DataProVe: A **Data** Protection Policy and System Architecture Verification Tool

Version 0.9.1

User manual

August 2020

Author:

VINH T. TA

Laboratory of Security and Forensic Research in Computing (SAFeR Lab.)

School of Psychology and Computer Science

University of Central Lancashire (UCLan)

## Introduction

DataProVe is a tool with GUI written in Python that allows a user to specify a high-level data protection (or privacy) policy and a system architecture, then verify the conformance between the specified architecture and the high-level policy. The main goal of the tool is to help a system designer at the higher level (compared to the other tools that mainly focus on the protocol level.), such as policy and architecture design. This step can be important to spot high-level design flaws early, before going ahead with the lower level system specification.

The verification engine of DataProVe is based on logic, combining both the so-called backward and forward search strategies.

For the theoretical foundation of the tool, please check the following paper(s):

[1] Vinh T Ta, *DataProVe: A Data Protection Policy and System Architecture Verification Tool*, pp. 1-36. 2020. [Link to the paper](#) (long version).

[2] Vinh T Ta, *Privacy by Design: On the Formal Design and Conformance Check of Personal Data Protection Policies and Architectures*, pp. 1-41. 2018. [Link to the paper](#) (long version).

The tool is available in two formats, .exe and .pyc. You just need to download and double click on them to run the program.

- You may be asked in Windows if you really want to run/trust the .exe file, because by default, any .exe extension is suspicious for the operating systems, especially for Windows. It will offer you an option to proceed though (the app contains no malicious code, so it's safe to run. This can be checked at <https://www.virustotal.com/gui/>).
- If you want to run the .pyc file (normally, you won't get warnings like the .exe case), then you need to install, ideally, Python version 3.8.2 or 3.8.5 32bit (or above, from <https://www.python.org/downloads/>). This file can be run by either double click or from command line, e.g., under Windows cmd using the command (assumed that you already added Python to path, see this guide <https://geek-university.com/python/add-python-to-the-windows-path/>):

```
python "<replace_with_path_to_file>\DataProVe-v0.9.pyc".
```

The template policy (.pol) and architecture (.arch) files can be found in the zip called "Template files used in the manual", just open them in the app and try.

Demo videos can be watched from [here](#).

Github link to the tools and templates/examples: <https://github.com/vinhgithub83/DataProVe> .

## The System Architecture Specification Page

After launching the tool, as depicted in Figure 1, the default page can be seen, where the user can specify the architecture.

The menu bar includes the main options “POLICY”, “ARCHITECTURE” and “VERIFY”, for policy specification, architecture specification and conformance verification between them. More specifically, under the option “POLICY”, the user can specify a new data protection policy, save the policy, and open a saved policy.



*Figure 1 After launching the app, the system architecture specification page can be seen.*

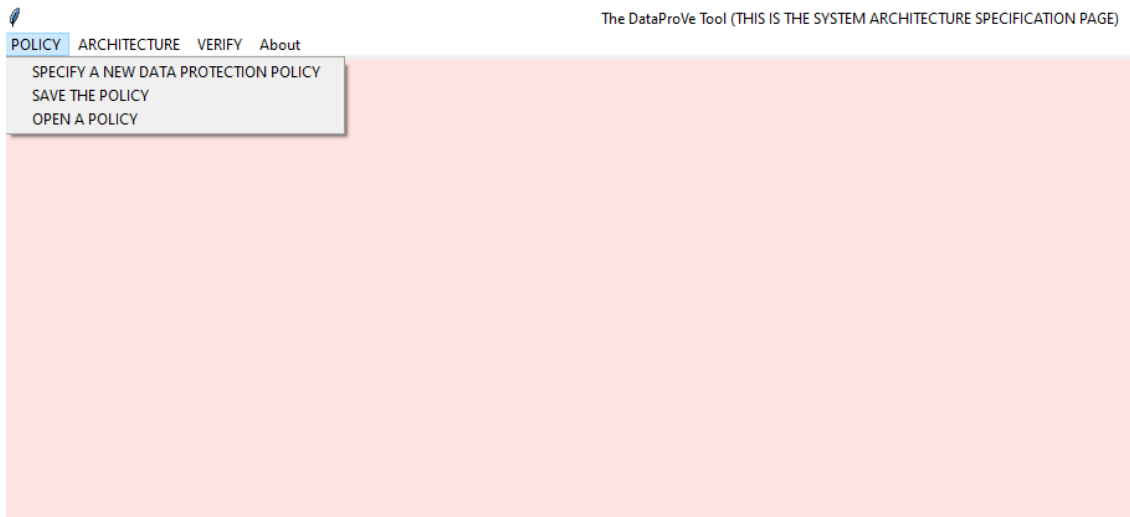


Figure 2.

Under the option “ARCHITECTURE”, the user can save an architecture, open a saved architecture, add an architectural component (main or sub-component), specify the relationship between the main and sub-components, and finally, save an architecture to a file.



Figure 3.

The main difference between the 1<sup>st</sup> option (SAVE THE ARCHITECTURE) and the last option (PRINT ARCHITECTURE TO FILE) is that while in the 1<sup>st</sup> case, all the GUI elements and the settings are saved, the last option only save the actions defined in an architecture.

Finally, the under the option “VERIFY”, we can verify the conformance between our specified policy and architecture.



Figure 4.

A policy can be saved in a file with an extension *\*.pol*, while an architecture is saved in *\*.arch* as can be seen in the pictures.

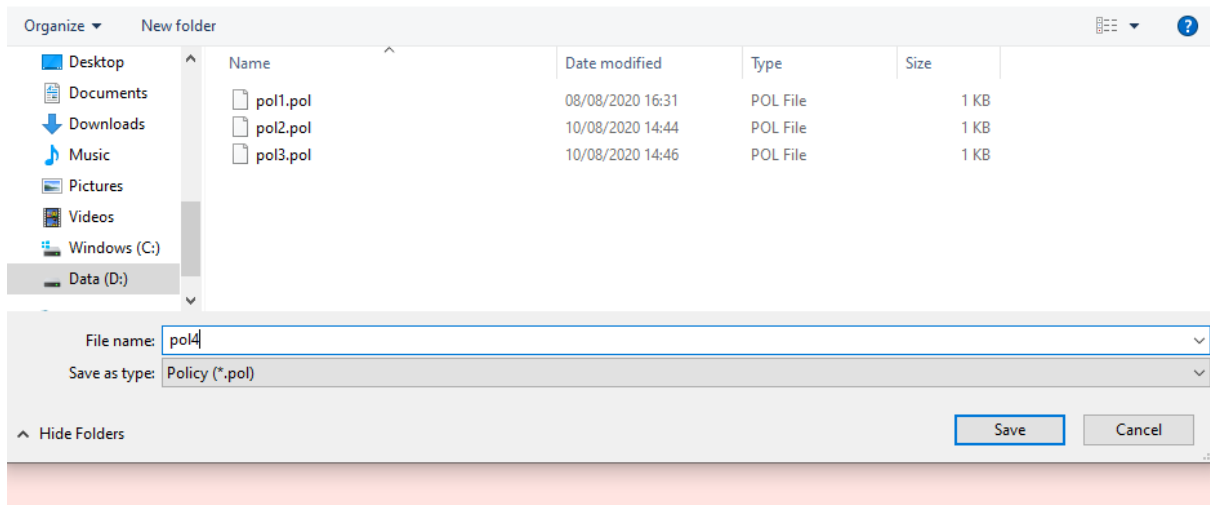


Figure 5.

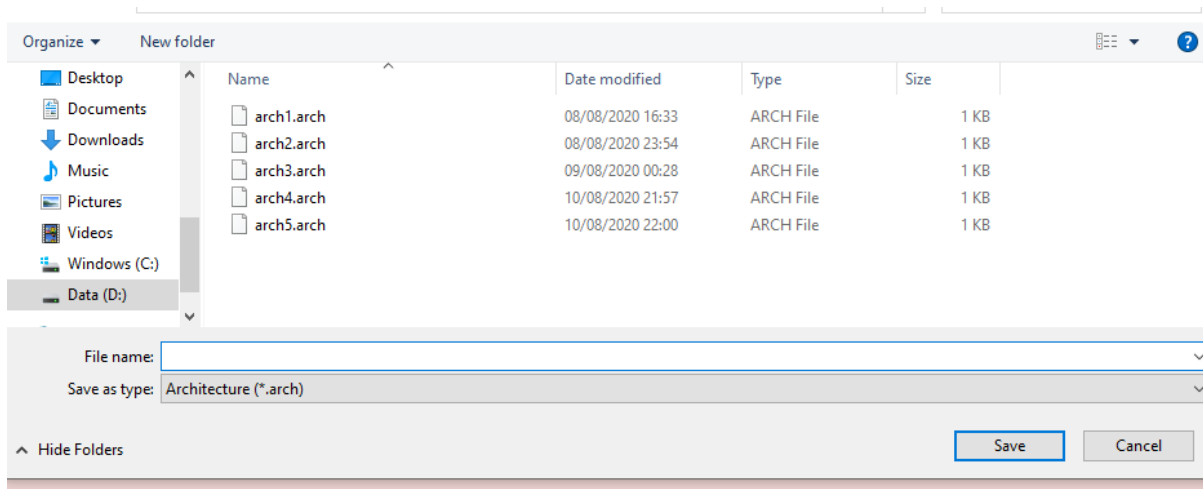


Figure 6.

The saved policies and architecture can be opened using the open policy and architecture options, respectively.

Under the option “ARCHITECTURE”, we can add the architectural components to the design (using the option “ADD A COMPONENT”).

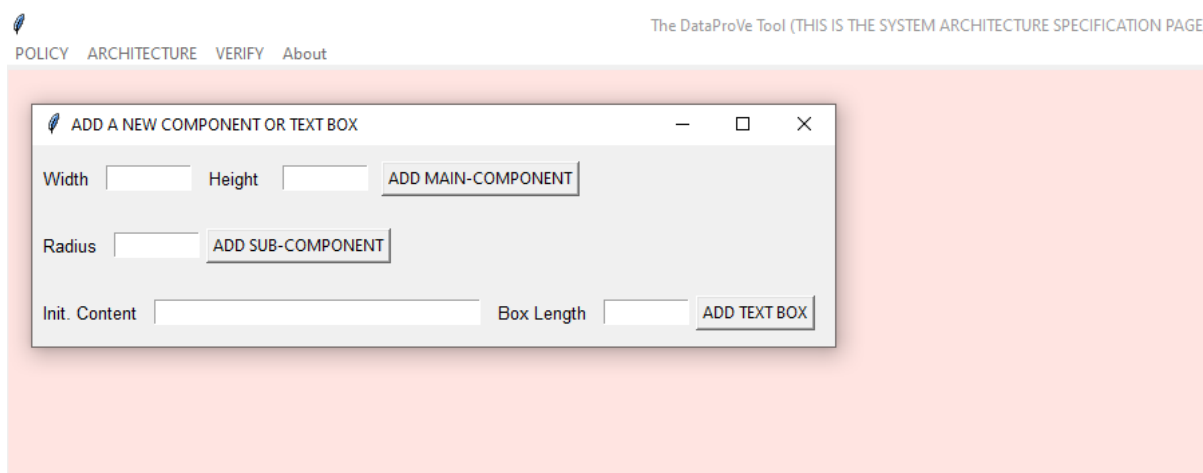


Figure 7.

DataProVe supports two types of components, the so-called main-components, and the sub-components. The main components can represent an entire organisation, system or entities that consists of several smaller components, such as a service provider, a customer, or authority (trusted third-party organisation). Sub-components are elements of a main component, for example, a service provider can have a server, a panel, or storage place. A main component usually has access to the data handled by its own sub-components, but this is not always the case, for instance, two main components can share a sub-component and only one main-component has access to its data. This can happen, for example, when a service provider operates a device of a trusted third party, but it does not have free access to the content of the data stored inside the device.

In this version of DataProVe, main components are represented by rectangular shapes, while sub-components are represented by circles. In order to add a new main components, the users can click on the bottom left button called “ADD MAIN COMPONENT”, but before that, they have to specify the size of the rectangle (the first text field is for the horizontal edge, while the second is for the vertical.). Examples can be seen in Figures 8-10.

In this document, we will interchange between the two terms *entity* and *component*, because the term entity has been used in our theoretical papers, while the tool uses the term component more. They refer to the same thing in our context.

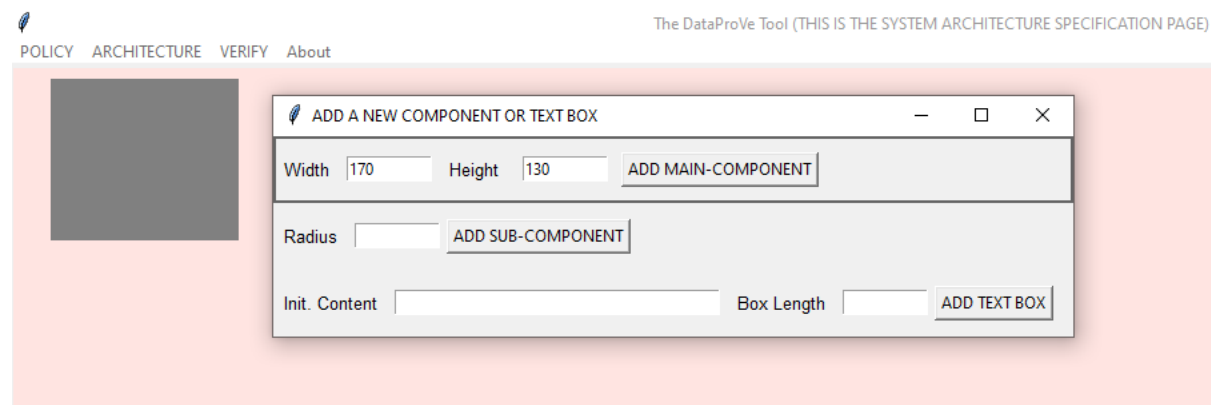


Figure 8. Adding a main component of size 170x130.

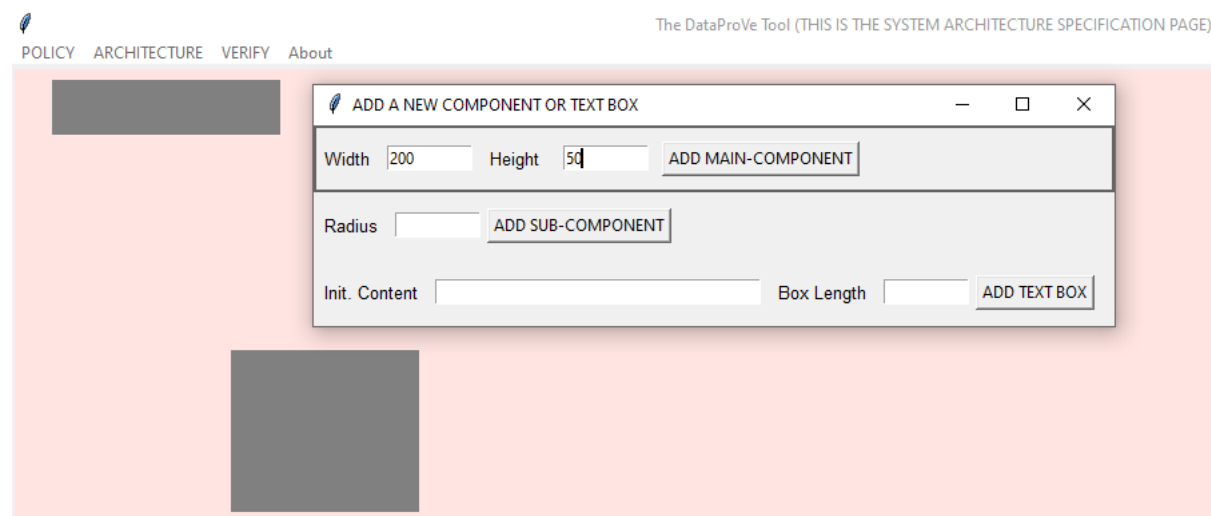


Figure 9. Adding a new main component of size 200x50.

The users can use the mouse (by holding the left-most button) to move an object from one location to another. To add a new sub-component, the user can provide the size of its radius and click on the button called “ADD SUB-COMPONENT”. Again, to move the sub-component from one place to another the user should hold the left mouse button and drag the object to the target place.

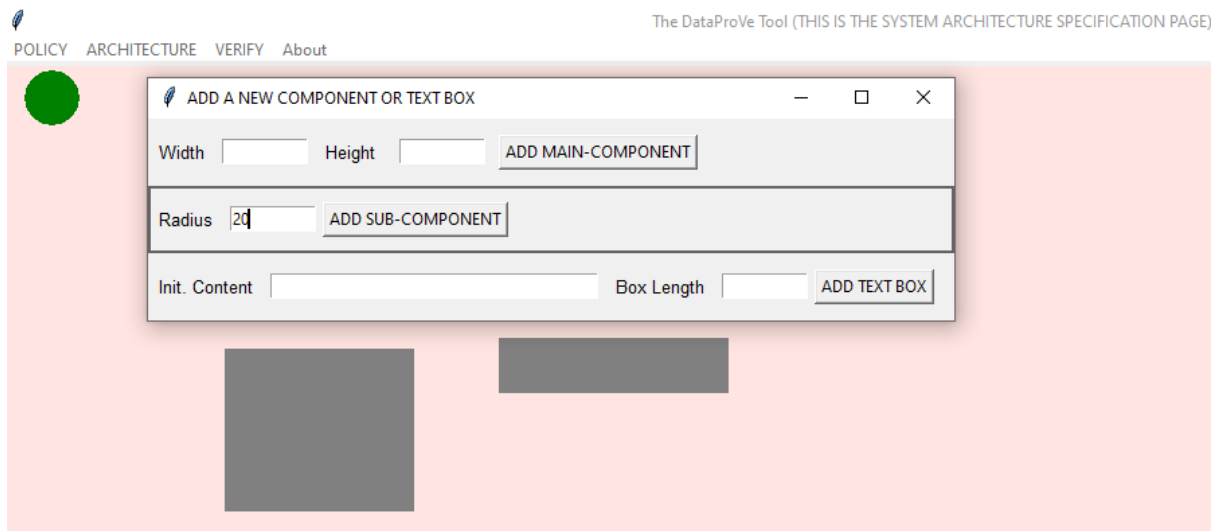


Figure 10. Adding a new sub-component with a radius size of 20.

To change the colour of an object, click once with the middle mouse button (if any), or the scroll wheel, then choose a preferable colour.

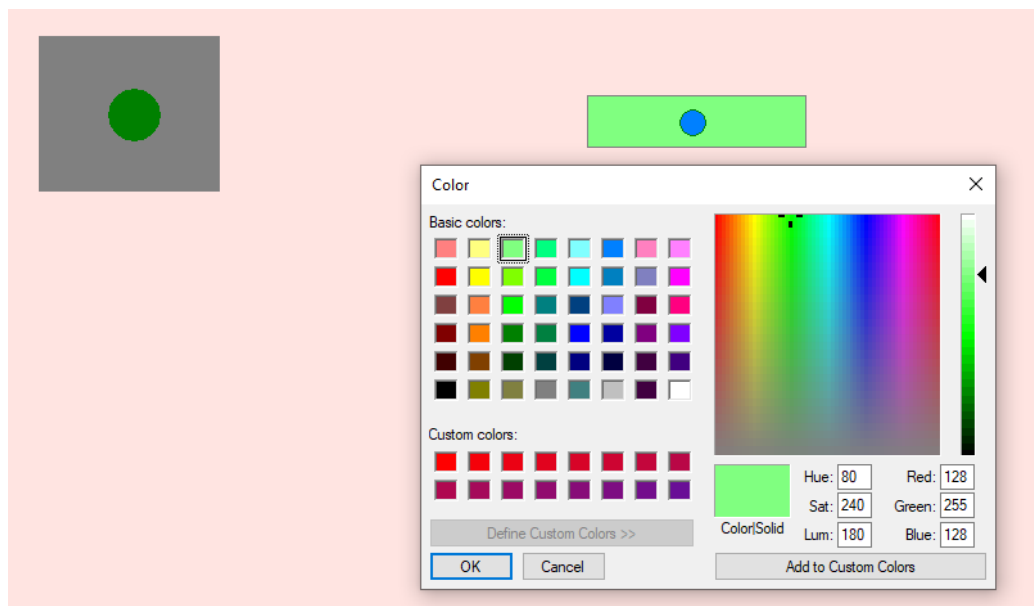


Figure 11. Change the colour of the objects (main- and sub-components).



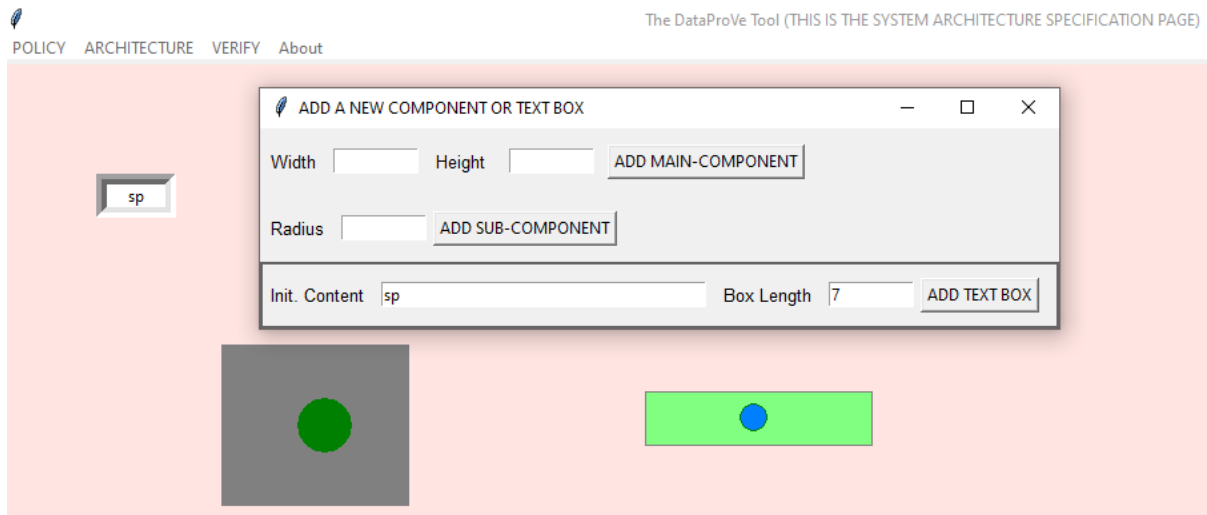


Figure 12. Add description (name) of the objects (sp, 7).

To add name of the main- and sub-components, the user can provide a name appears in the textbox and the size of the textbox, for instance, the textbox with the name “sp” (referring to a service provider) of size 7 (as shown in Figure 12.).



Figure 13. Assigned names for each main and sub-component.

As depicted in Figure 13, in this example we created two main components, a service provider that has a server, and the client that has a meter (for smart meter reading) installed at its place.

In order to specify which main component can have access to which sub-component, the user can click on the button called “SPECIFY THE RELATIONSHIP BETWEEN THE MAIN AND SUB COMPONENTS”, this will open a window where we can give which main component has access to the data in which sub-component. An example can be seen in Figure 14, between sp, server and meter.

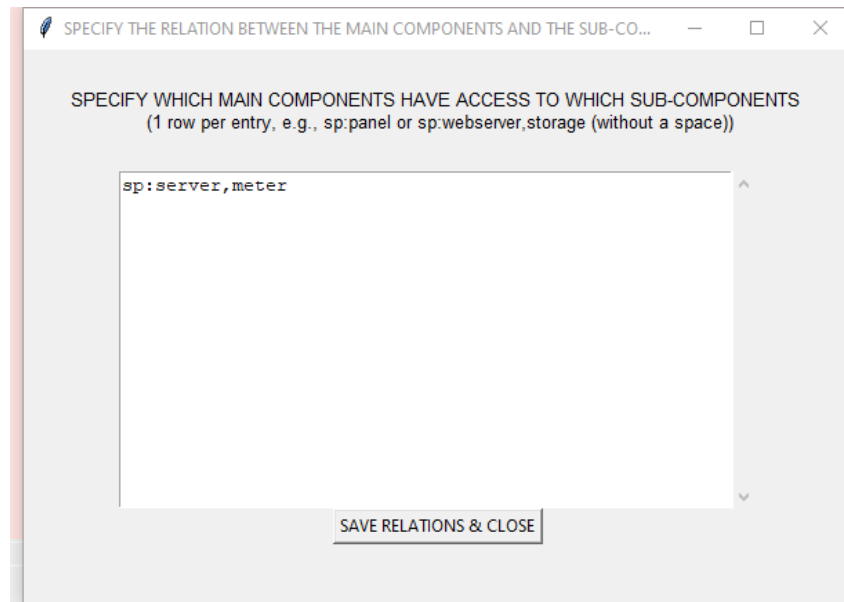


Figure 14. Specify which main component has access to the data in which sub-component (sp has access to server and meter).

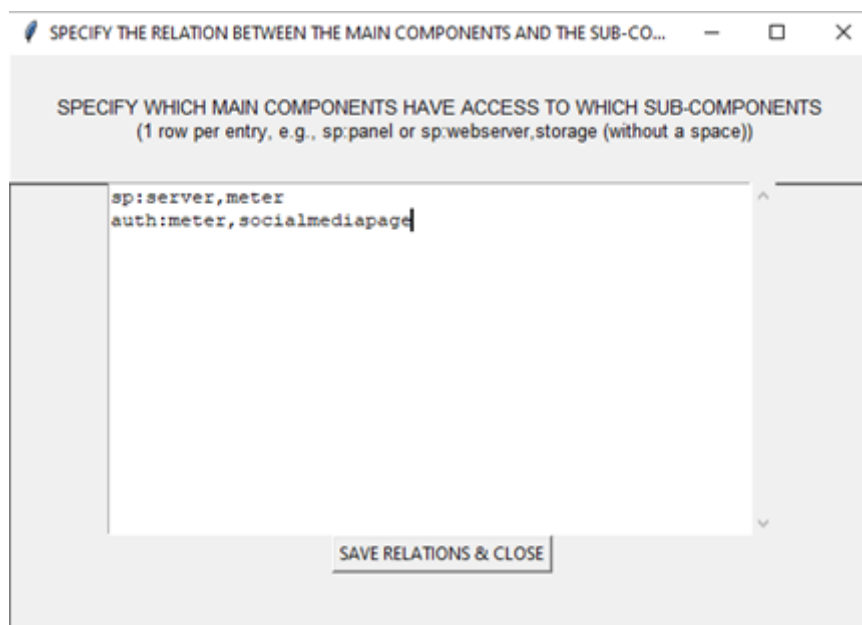


Figure 15. sp (service provider) has access to server and meter, auth (third party authority) has access to meter and a social media webpage.

Note that any component can be on the right-side of a relationship above, not only the sub-components of a component. For example, it can be a public profile page of a social media site.

Finally, to illustrate which component receives which message, in DataProVe, one can draw an arrow from one component to another component. To draw an arrow, the users need to hold the right mouse button and drag from one component to another (a component can be either main or sub).

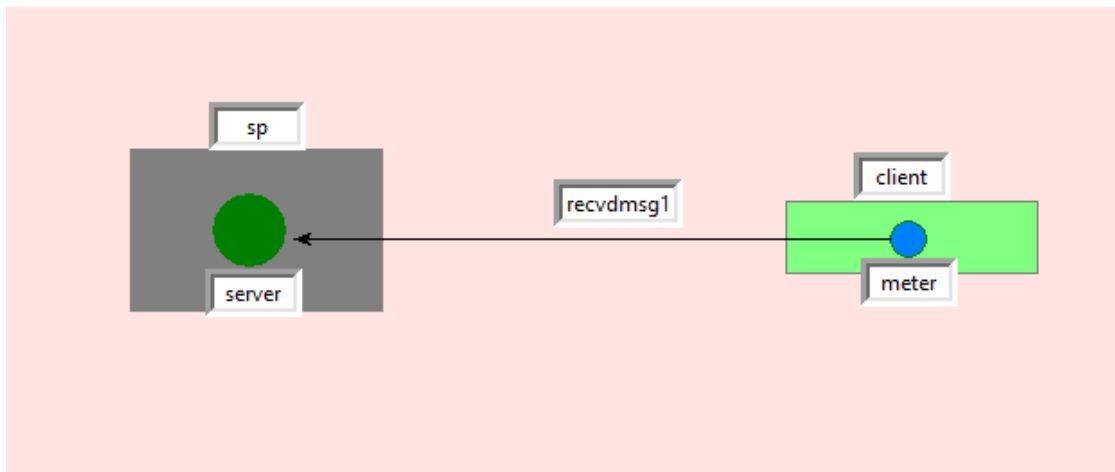


Figure 16. Draw an arrow from the component meter to server.

In Figure 10, a new text box is created with the name recvdmsg1, which denotes that the server receives the message msg1. To specify the content of msg1, double click on the text box. For instance, in Figure 11, the content says that sp can receive a reading that contains the energy consumption (energy) and the customer ID (custID).

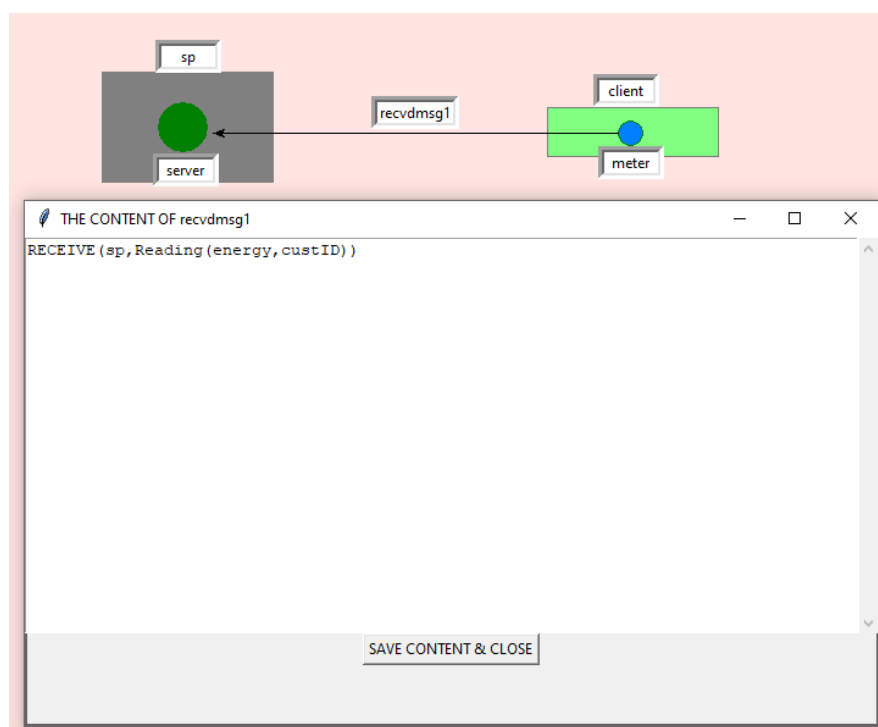


Figure 17. Specify the message content of recvdmsg1.

To delete an object (a component, text box, or an arrow), the user should double click on the given object.

<b>GUI COMMANDS</b>	<b>Effect</b>
Hold the leftmost mouse button and drag.	Move an object from one place to another.
1 click on the middle mouse button/mouse wheel.	Change the colour for main -, sub-component or an arrow.
Using rightmost button of the mouse: 1 click on the start point and 1 click on the end point.	Draw a line/arrow from the chosen start point to the end point.
Double click with the leftmost mouse button on an object (components, arrow, textbox).	Delete a given object.
Scroll the mouse wheel.	Zoom in, zoom out.

In the architecture level, we distinguish entity/component, actions and data, where actions specify what a component/entity can do on a piece of data (it may not perform this action eventually during a low-level system run, but there are instances of the system run that where this action happens).

#### ACTIONS:

Actions are words/string of all capital letters, and DataProVe supports the actions "OWN", "RECEIVE", "RECEIVEAT", "CREATE", "CREATEAT", "CALCULATE", "CALCULATEAT", "STORE", "STOREAT", "DELETE", "DELETEWITHIN".

The syntax of each action is as follows.

*Note: no space character is allowed when specifying the actions in the bullet points below. In addition, in this version, Datatypes cannot be a tuple (i.e., (data1,data2,...,data\_n ) ).*

The reserved/pre-defined keywords are highlighted in bold, while the non-bold text can be freely defined by the user:

- **OWN**(component,Datatype) :  
The action defines that a component (e.g., sp, auth, server, meter etc.) can own a piece of data of type Datatype. For example, **OWN**(server,spkey) say they server can own the a piece of data of type service provider key (spkey).
- **RECEIVE**(component,Datatype):  
This action defines that a component can receive a piece of data of type Datatype, for example, **RECEIVE**(server,Sicknessrecord(name,insurancenumner)) says that server can receive a sickness record that contains a piece of data of type name and insurance number.
- **RECEIVEAT**(component,Datatype,**Time(t)**):  
This action is similar to the previous one, except that here we also need to define the time when the data can be received. Since at the architecture level we do not intent to go into specifying the concrete time value, the keywords/generic time construct **Time(t)** specifies that *component* can receive a piece of data of type Datatype at some (not specific) time t.  
RECEVEAT is used to define when a consent (**Cconsent**(Datatype), **Uconsent**(Datatype), **Sconsent**(Datatype), **Fwconsent**(Datatype)) is received.
- **CREATE**(component,Datatype):  
This action defines that a component can create a piece of data of type Datatype, for instance, **CREATE**(sp,Account(name,address,phone)) defines that a service provider sp can create an account that contains three pieces of data of types name, address and phone number.
- **CREATEAT**(component,Datatype,**Time(t)**):  
This action defines that a component can create a piece of data of type Datatype at some (not specific) time t. For example, **CREATE**(sp,Account(name,address,phone),**Time(t)**).
- **CALCULATE**(component,Datatype):

This action defines that a component can calculate a piece of data of type Datatype, for instance, **CALCULATE**(sp,Bill(energyconsumption)) defines that a service provider sp can calculate a bill using a piece of data of type energy consumption.

- **CALCULATEAT**(component,Datatype,**Time(t)**):

This action defines that a component can calculate a piece of data of type Datatype at some (not specific) time t. For example, **CALCULATE**(sp,Bill(energyconsumption),**Time(t)**).

- **STORE**(storageplace,Datatype):

This action defines that a service provider can store a piece of data of type Datatype in storageplace, where storageplace can be **mainstorage**, **backupstorage**. These reserved keywords define a collection of storage place(s) that can be seen as “main” storage, or “backup” storage of a **service provider**, respectively.

For example, **STORE**(**mainstorage**,Account(name,address,phone)) defines that a service provider can store an account that contains name, address and phone number in its main storage place(s).

- **STOREAT**(storageplace,Datatype,**Time(t)**):

This action defines that a component can store a piece of data of type Datatype in the place(s) storageplace at some (not specific) time t.

For example, **STORE**(**mainstorage**,Account(name,address,phone),**Time(t)**) defines that an account with a name, address and phone number can be stored in the main storage of the service provider at some time t.

- **DELETE**(storageplace,Datatype):

The action delete is closely related to the action store, as it defines that a piece of data of type Datatype can be deleted from storageplace.

For example, **DELETE**(**mainstorage**,Account(name,address,phone)) captures that a service provider can.

- **DELETEWITHIN**(storageplace,Datatype,**Time(tvalue)**):

This action captures that once the data is stored, a component must delete a piece of data of type Datatype within the given time value tvalue (tvalue is a data type for time values). Unlike the non-specific **Time(t)**, which is a predefined construct, tvalue is defined by the user, and takes specific time values such as 3 years or 2 years 6 months.

For example, **DELETE**(**mainstorage**,Account(name,address,phone),**Time(2y)**) defines that the service provider must delete an account from its main storage within 2 years.

## COMPONENTS/ENTITY:

A component can be specified by a string of all lower case, for example, a service provider can be specified by `sp`, or a third-party authority by `auth` (obviously they can be specified with any other string).

DataProVe supports some pre-defined or reserved components/entities, such as **sp**, **trusted**, **mainstorage**, **backupstorage**.

- **sp**: this reserved keyword defines a service provider. DataProVe only allows single service provider at the same time (in the specification of a policy and architecture).
- **trusted**: this reserved keyword defines a trusted authority that is able to link a pseudonym to the corresponding real name.
- **mainstorage**: this reserved keyword defines the collection of main storage places of a service provider.
- **backupstorage**: this reserved keyword defines the collection of backup storage places of a service provider.

Note: An entity/component is always defined as *the first argument* of an action.

## DATA TYPES:

DataProVe supports two groups of data types, the so-called compound data types, and simple data types.

- **Simple data types** do not have any arguments, and they are specified by strings of all lower cases, without any space or special character. Example simple data types include `name`, `address`, `phonenumber`, `nhsnumber`, etc.
- **Compound data types** have arguments, and they are specified by strings that start with a capital letter followed by lower cases (again without any space or special character). For example, `Account(name,address,phone)` is a compound data type that contains three simple data types as arguments. Another example compound data type can be `Hospitalrecord(name,address,insurance)`. Any similar compound data types can be defined by the user. *We note that the space character is not allowed in the compound data types.*

Nested compound data types are compound data types that contain another compound data types. For instance, `Hospitalrec(Sicknessrec(name,disease),address,insurance)` captures a hospital record that contains a sickness record of a name and disease, and an address, and finally, an insurance number.

Note: This version of DataProve supports three layers of nested data types.

DataProVe has pre-defined or reserved data types, such as

- The types of consents: **Cconsent**(Datatype), **Uconsent**(Datatype), **Sconsent**(Datatype), **Fwconsent**(Datatype).

We do not differentiate among the different consent format, it can be e.g. written consent, or online consent form, or some other formats.

- **Cconsent**(Datatype): This is a type of collection consent on a piece of data of type Datatype. For example, **Cconsent**(illness), **Cconsent**(Account(creditcard,address)) capture the collection consent on the illness information, and the account containing a credit card number and address.
  - **Uconsent**(Datatype): A type of usage consent on a piece of data of type Datatype. For example, **Uconsent**(Energy(gas,water,electricity)), **Uconsent**(address).
  - **Sconsent**(Datatype): A type of storage consent on a piece of data of type Datatype. For example, **Sconsent**(personalinfo), **Sconsent**(Account(creditcard,address)) defines the types of storage consent on a type of personal information and account, respectively.
  - **Fwconsent**(Datatype,component): A type of forward/transfer consent on a piece of data of type Datatype, and a component to whom the data is forwarded/transferred. E.g. **Fwconsent**(personalinfo,auth), **Fwconsent**(Account(creditcard,address),auth) defines the type of forward consent on the type of personal information and account, respectively, as well as a third party authority (auth) to which the given data is forwarded.
- The types of time and time value: **Time(t)** or **Time(tvalue)**, where **Time()** is a time data type, while the pre-defined special keyword **t** denotes a type of non-specific time, and tvalue is a type of time value (such as 5 years, 2 hours, 1 minute, etc.).

tvalue is a (recursive) type and takes the form of

$$tvalue ::= y \mid mo \mid w \mid d \mid h \mid m \mid numtvalue \mid tvalue + tvalue$$

where y specifies a year, mo a month, w a week, d a day, h an hour and m a minute. Further, numtvalue is the a number (num) before tvalue, for example if num = 3 and tvalue = y, then numtvalue is 3y (i.e. 3 years). Additional examples include tvalue = 5y + 2mo + 1d + 5m.

It is important to note that **Time(tvalue)** can only be used in the action **DELETEWITHIN**. **RECEIVEAT**, **CREATEAT**, **CALCULATEAT**, **STOREAT** must contain the non-specific time **Time(t)**.

For example, the actions

- **DELETEWITHIN**(sp,mainstorage,Webpage(photo,job),**Time**(10y+6mo))  
Any webpage must be deleted from the main storage of the service provider within 10 years and 6 months.



- **RECEIVEAT**(sp,Cconsent(illness),Time(t))  
The service provider can receive a collection consent on illness information at some non-specific time *t*.
- **RECEIVEAT**(sp,Uconsent(Webpage(photo,job)),Time(t))  
The service provider can receive a usage consent on a webpage at some non-specific time *t*.
- **STOREAT**(backupstorage,Webpage(photo,job),Time(t))  
The service provider can store a webpage in its back up storage places at some non-specific time *t*.
- **CREATEAT**(server,Account(name,address),Time(t)):  
The service provider can create an account that contains a name and address in at some non-specific time *t*.
- **CALCULATEAT**(sp,Bill(tariff,Energy(gas,water,electricity)),Time(t)):  
The service provider can create an account that contains a name and address in at some non-specific time *t*.
- The type of metadata and meta values: **Meta**(Datatype)  
This data type defines the type of metadata located in the header of the packets, the meta information often travels through a network without any encryption or protection, which may pose privacy concern. Careful policy and system design are necessary to avoid privacy breach caused by the analysis of metadata.

Note: **Meta**(Datatype) is always defined as the *last argument* in a piece of data.

Example application of metadata include:

- **RECEIVE**(sp,Sicknessrec(name,disease,Meta(ip))):  
This action defines that the service provider can receive a packet that containing a name and disease, but the packet also includes the metadata IP address of the sender computer. We note that this syntax is simplified in terms that it aims to eliminate the complexity of nested data type. Specifically, this syntax abstracts away from the definition of the so-called packet data type, an “abbreviation” of the lengthy **RECEIVE**(sp,Packet(Sicknessrec(name,disease),Meta(ip))).
- **RECEIVE**(sp,Sicknessrec(name,disease,Meta(Enc(ip,k)))):  
This action is similar to the previous one, but now the metadata IP address is encrypted with a key *k*.
- **RECEIVEAT**(sp,Sicknessrec(name,disease,Meta(ip)),Time(t)):

This action is similar to the first one, but it includes the time data types at the end. It defines that the service provider receives the sickness record along with the IP address of the sender device, at some non-specific time  $t$ .

Obviously, any metadata can be defined instead of IP address in the examples above.

- The type pseudonymous data: **P(Datatype | component)**  
This data type defines the type of pseudonymous data, for example, a pseudonym. The argument can either a data type or a component. Pseudonym is a means for achieving a certain degree of privacy in practice as the real identity/name and the pseudonym can only be linked by a so-called trusted authority. DataProVe also captures this property, namely, only the component **trusted** can link the pseudonym to the real name/identity.

For example,

- **RECEIVE**(sp,Sicknessrec(**P**(name),disease))  
This action defines that a service provider can receive a sickness record, but this time, the name in the record is not the real name but a pseudonym, hence, the service provider cannot link a real name to a disease.
- **RECEIVE**(**trusted**,Sicknessrec(**P**(name),disease))  
This is similar to previous case, but the trusted authority can receive a sickness record instead of the service provider.
- **RECEIVE**(sp,Sicknessrec(**P**(name),disease,**Meta**(ip))):  
Again, this is similar to the first case, but with metadata.
- **RECEIVEAT**(sp,Sicknessrec(name,disease,**Meta**(ip)),**Time**(t)):  
This is similar to previous case, but also include the time data type.

- The types of cryptographic primitives and operations: DataProVe supports the basic cryptographic primitives for the architecture. Again, we provide the reserved keywords in bold.
  - Private key: **Sk**(Pkeytype)  
This data type defines the type of private key used in asymmetric encryption algorithms. Its argument has a type of public key (Pkeytype). We note that public key is not a reserved data type.
  - Symmetric encryption: **Senc**(Datatype,Keytype)  
This is the type of the cipher text resulted from a symmetric encryption, and has two arguments, a piece of data and a symmetric key (Keytype).  
For example,
    - **RECEIVE**(sp,**Senc**(Account(name,address),key))  
This specifies that a service provider can receive a symmetric key encryption of an account using a key of type key.
    - **RECEIVE**(sp,**Senc**(Account(**Senc**(name,key),address),key))

This specifies that a service provider can receive a symmetric key encryption of an account that contains another encryption of a name, using a key of type key.

- **OWN(sp,key)**

This specifies that a service provider can own a key of type key.

- Asymmetric encryption: **Aenc(Datatype,Pkeytype)**

This is the type of the cipher text resulted from an asymmetric encryption, and has two arguments, a piece of data and a public key (Pkeytype).

For example,

- **RECEIVE(sp,Aenc(Account(name,address),pkey))**

This specifies that a service provider can receive an asymmetric key encryption of an account using a public key of type pkey.

- **CALCULATE(sp,Sk(pkey))**

This specifies that a service provider can calculate a private key corresponding to the public key (of type pkey).

- **OWN(sp,pkey)**

This specifies that a service provider can own a public key of type pkey.

- Message authentication code (MAC): **Mac(Datatype,Keytype)**

This is the type of the message authentication code that has two arguments, a piece of data and a symmetric key (Keytype).

For example,

- **RECEIVE(sp,Mac(Account(name,address),key))**

This specifies that a service provider can receive a message authentication code of an account using a key of type key.

- Cryptographic hash: **Hash(Datatype)**

This is the type of the cryptographic hash that has only one argument, a piece of data.

For example,

- **RECEIVE(server,Hash(password))**

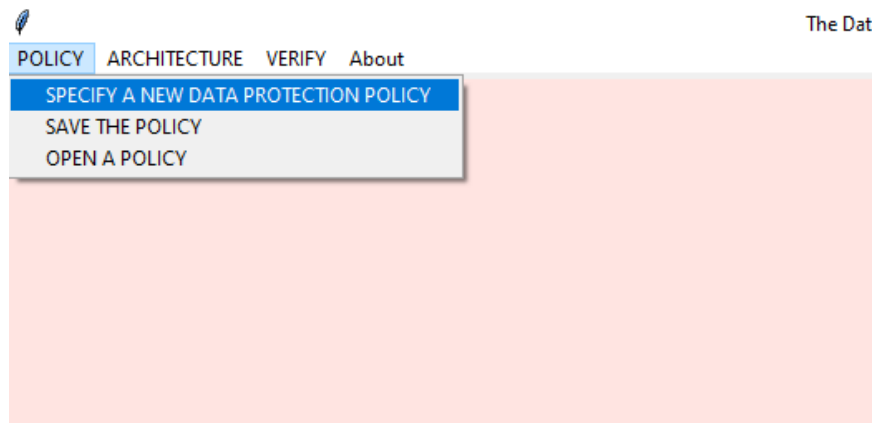
This specifies that a server can receive a hash of a password.

- **STORE(mainstorage,Hash(password))**

This specifies that a service provider can store a hash of a password in its main storage place(s).

## The Data Protection Policy Specification Page

To define a high-level data protection policy, from the architecture page, on the top left corner we choose the menu called “POLICY”, then click on “SPECIFY A NEW DATA PROTECTION POLICY”.



### Entities/Components (the top part):

The policy page has three parts, the top part is to specify the entities/components in the system, such as authority, client etc. On the left side, the user is expected to provide a short notation, and on the right side, the full name/description to help identifying the meaning of the notation. For instance, in Figure 18, the notation is *auth*, and the description is *third party authority*. After adding a new entity, it will appear in the drop-down option menu in the bottom part. Note that **the entity *sp* (service**

A screenshot of the 'The DataProVe Tool (THIS IS THE DATA PROTECTION POLICY SPECIFICATION PAGE)'. The page is divided into three main horizontal sections. The top section is a dark blue header. The middle section is a light blue area containing a form. The bottom section is a light blue footer. The form in the middle section has three main parts: 1. 'PROVIDE A NEW ENTITY:' with a text input field containing 'auth'. 2. 'PROVIDE A DESCRIPTION:' with a text input field containing 'third party authority'. 3. An 'ADD NEW ENTITY' button. Below these, there is a section for 'PROVIDE A GROUP OF DATA TYPES:' with a text input field, 'IS THIS UNIQUE?' with a 'No' button, and 'THE DATA TYPES IN THIS GROUP:' with a large empty box and an 'ADD DATA GROUP & TYPES' button. At the bottom, there is a row of buttons: 'Choose an entity' (with a dropdown showing 'sp (service provider)'), 'Choose a data group' (with a dropdown), 'Choose a data type' (with a dropdown), and 'SPECIFY EACH SUB-POLICY:' followed by a row of buttons: 'Data Collection', 'Data Usage', 'Data Storage', 'Data Retention', 'Data Transfer', 'Data Possession', 'Data Connection Permitted', and 'Data Connection Forbidden'.

Figure 18. The Policy Specification Page.

**provider**) is a **pre-defined entity** that is already added by default (hence, the user does not need to add). The user can specify any other entities.

#### Data groups/Data types (the middle part):

The middle part in the policy specification page is for defining the data groups and data types. As shown in the figures below, on the left side (PROVIDE A GROUP OF DATA TYPES), the user can define a group of data types, for instance, in Figure 19, a data group denoted by *personalinfo* is defined which includes four data type, name, address, dateofbirth, and phonenumber.

The screenshot shows a form with a light blue background. On the left, under the label "PROVIDE A GROUP OF DATA TYPES:", there is a text input field containing "personalinfo". To its right is a section labeled "IS THIS UNIQUE?" with two radio buttons: "Yes" (which is selected) and "No". Further right, under the label "THE DATA TYPES IN THIS GROUP:", there is a text area containing the following text: "name", "address", "dateofbirth", and "phonenumber". On the far right, there is a button labeled "ADD DATA GROUP & TYPES".

Figure 19. Specifying data groups (*personalinfo*) and its data types.

The option menu in the middle (called "IS THIS UNIQUE") expects the user to provide if the data group together with its data types can be used to *uniquely identify* an individual. For instance, a name alone cannot be used to unique identify an individual, but a name together with an address, date of birth and phone number, can be, so the option "Yes" was chosen. Another example is shown in Figure 20, with the data group called *energy* (refers to energy consumption) and its data types, gas, water, and electricity consumption. This type group together with its types cannot be used to uniquely identify an individual, so the option "No" was chosen.

The screenshot shows a form with a light blue background. On the left, under the label "PROVIDE A GROUP OF DATA TYPES:", there is a text input field containing "energy". To its right is a section labeled "IS THIS UNIQUE?" with two radio buttons: "No" (which is selected) and "Yes". Further right, under the label "THE DATA TYPES IN THIS GROUP:", there is a text area containing the following text: "gas", "water", and "electricity". On the far right, there is a button labeled "ADD DATA GROUP & TYPES".

Figure 20. Specifying data groups (*energy*) and its data types.

The user can also provide data types in the field "PROVIDE A GROUP OF DATA TYPES", but this case the field "DATA TYPES IN THIS GROUP" should be left empty. DataProVe supports data groups for convenience purposes, as in case of complex systems there can be a huge number of data types, and manually defining a data protection policy for each data type can be inconvenience.

The screenshot shows a form with a light blue background. On the left, under the label "PROVIDE A GROUP OF DATA TYPES:", there is a text input field containing "name". To its right is a section labeled "IS THIS UNIQUE?" with two radio buttons: "No" (which is selected) and "Yes". Further right, under the label "THE DATA TYPES IN THIS GROUP:", there is an empty text area. On the far right, there is a button labeled "ADD DATA GROUP & TYPES".

Figure 21. Specifying only data type (*name*) that is non-unique.

For instance, in Figure 21, we only define a data type called name, which cannot be used to uniquely identify an individual. We left the text box on the right empty. In Figure 22, a data type of insurance number, insurancenumbe, is specified that can be used to uniquely identify an individual.

Figure 22 shows a form interface for specifying data types. It includes a section labeled "PROVIDE A GROUP OF DATA TYPES:" with a text input field containing "insurancenumbe". To the right of this field is a checkbox labeled "IS THIS UNIQUE?" which is checked, with a "Yes" label. Further right is a large empty text box labeled "THE DATA TYPES IN THIS GROUP:". On the far right is a button labeled "ADD DATA GROUP & TYPES".

Figure 22. Specifying only data type (insurance number) that is unique.

## Policy specification (the bottom part):

A data protection policy is defined on a data group/type and an entity. In DataProVe, each policy consists of eight sub-policies, to achieve a fine-grained requirement specification. The users do not have to define all the eight sub-policies, but they can if it is necessary.

Figure 17 shows the "Policy Specification Page (entities)". It features three dropdown menus: "Choose an entity" with options "auth (third party authority)", "sp (service provider)", and "auth (third party authority)"; "Choose a data group" with a default value of "-"; and "Choose a data type" with a default value of "-". To the right is a section labeled "SPECIFY EACH SUB-POLICY:" with eight checkboxes: "Data Collection", "Data Usage", "Data Storage", "Data Retention", "Data Transfer", "Data Possession", "Data Connection Permitted", and "Data Connection Forbidden".

Figure 17. The Policy Specification Page (entities).

Figure 23 shows the "Policy Specification Page (data groups)". It features three dropdown menus: "Choose an entity" with a default value of "sp (service provider)"; "Choose a data group" with a dropdown menu showing "energy" and "personalinfo energy"; and "Choose a data type" with a dropdown menu showing "name". To the right is a section labeled "SPECIFY EACH SUB-POLICY:" with eight checkboxes: "Data Collection", "Data Usage", "Data Storage", "Data Retention", "Data Transfer", "Data Possession", "Data Connection Permitted", and "Data Connection Forbidden".

Figure 23. The Policy Specification Page (data groups).

The first five sub-policies (collection..., transfer) are defined only from the service provider's perspective. For the rest three sub-policies (data possession and connection), the user needs to choose from the drop-down option menus which entity/component, and data groups (exclusive) or data type the policy is defined for. If the user chooses a data group, then they should leave the data type option as the default "-", otherwise, if a data type is chosen then the data group should be left as the default (empty) value (as shown in Figure 23).

Choose an entity:

Choose a data group:

Choose a data type:

phonenumber  
water  
electricity  
address  
gas  
dateofbirth  
name

SPECIFY EACH SUB-POLICY :

Data Collection	Data Usage	Data Storage	Data Retention	Data Transfer	Data Possession	Data Connection Permitted	Data Connection Forbidden
-----------------	------------	--------------	----------------	---------------	-----------------	---------------------------	---------------------------

Figure 24. The Policy Specification Page.

The eight sub-policies are data collection, data usage, data storage, data retention, data transfer, data possession and the two data connection sub-policies.

### The data collection sub-policy

In the data collection sub-policy window, for a given entity and data group the user can specify whether consent is required to be collection when the selected entity collect a selected data group (Y for Yes/N for No), and then specify the collection purposes.

DATA COLLECTION POLICY

Does sp (service provider) collect consent before collecting the data (of type) personalinfo ? (give Y or N)

Collection Purposes (1 per row in the following format:  
action:data1,data2,..., e.g., create:account)

SAVE COLLECTION POLICY & CLOSE

Figure 25. The data collection sub-policy.

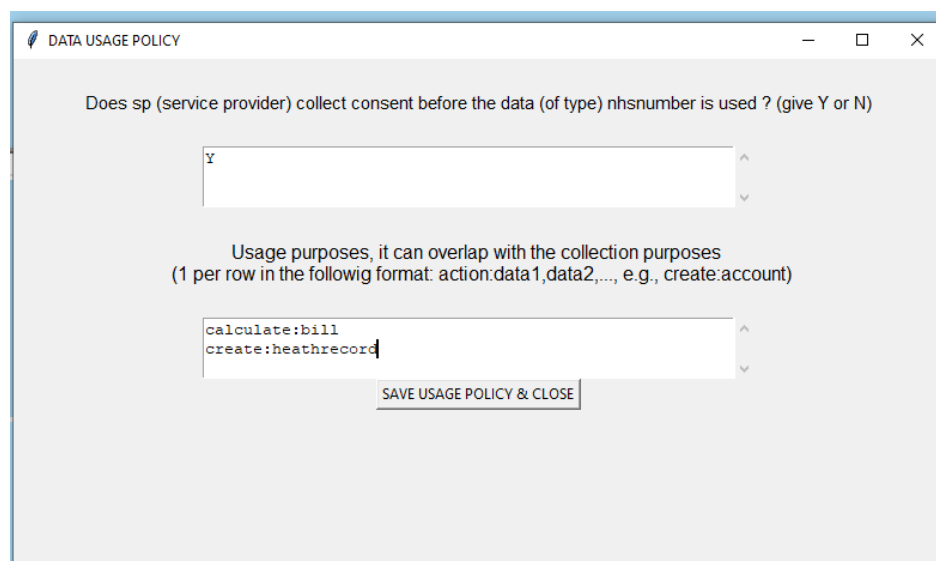
The collection purposes can be given row by row, each row with a different action in the format of:

***action1:data1,data2,...,data\_n***

where *action1* can be any action, while *data1, ..., data\_n* are compound data types (note that these compound data types do not need to be specified/added in the policy). For example, in Figure 25, the user sets that consent is required to be collected when the service provider collects the personal information. Then, the collection purpose for personal information is to create an account. The compound type account does not need to be defined in the policy.

#### *The data usage sub-policy*

The usage sub-policy is similar to the collection sub-policy but related to the consent and purposes for data usage. For instance, in Figure 26, a consent collection is required before using the data type NHS (national health service in UK) number, and the usage purpose of the NHS number is calculating a bill and creating a health record that contains the NHS number.



The screenshot shows a window titled "DATA USAGE POLICY" with standard window controls. Inside, there is a question: "Does sp (service provider) collect consent before the data (of type) nhsnumber is used ? (give Y or N)". Below this is a text input field containing the letter "Y". Further down, a label reads "Usage purposes, it can overlap with the collection purposes (1 per row in the followig format: action:data1,data2,..., e.g., create:account)". Below this label is a text area containing two lines of text: "calculate:bill" and "create:heathrecord". At the bottom right of the form area is a button labeled "SAVE USAGE POLICY & CLOSE".

Figure 26. The data usage sub-policy.

#### *The data storage sub-policy*

The data storage sub-policy is for specifying how an entity stores a selected data group. Similar to the previous two sub-policies one can specify if consent is required when/before a piece of data is stored (Y for Yes/N for No).



DATA STORAGE POLICY

Does sp (service provider) need to collect consent before the data (of type) energy is stored ? (give Y or N)

Y

Choose a Storage Option (How the Data (of Type) energy Will Be Stored By the Entity sp (service provider) )

Service Provider (Main & Backup Storage)

Service Provider (Main & Backup Storage)

Service Provider (Only Main Storage)

Not Service Provider (e.g., Decentralised or Client-side)

SAVE STORAGE POLICY & CLOSE

Figure 27. The data storage sub-policy.

The storage sub-policy offers four pre-defined options, “Service Provider (Main & Backup Storage)”, “Service Provider (Only Main Storage)”, “Not Service Provider”. The first option means that the data will be stored at the service providers’ own servers, in both main and backup storages. The second option is similar to the previous option, but the data is only stored in the main storage places, without back up. Finally, the third option is when the data is stored at the client devices or decentralised storage places that are not under control of the service provider.

#### The data transfer sub-policy

The sub-policy defines how a piece of data of a data type/group is transferred or forwarded. Again, the user needs to specify whether the consent is required to be collected when/before the data is transferred. Afterwards, entities can be added to whom the data will be transferred.

DATA TRANSFER POLICY

Does sp (service provider) need to collect consent before transferring the data (of type) energy ? (give Y or N)

Y

Choose to whom sp (service provider) will transfer the data (of type) energy for which consent needs to be collected before transferred

auth (trusted third-party)

auth

sp (service provider)

client (customer)

auth (trusted third-party)

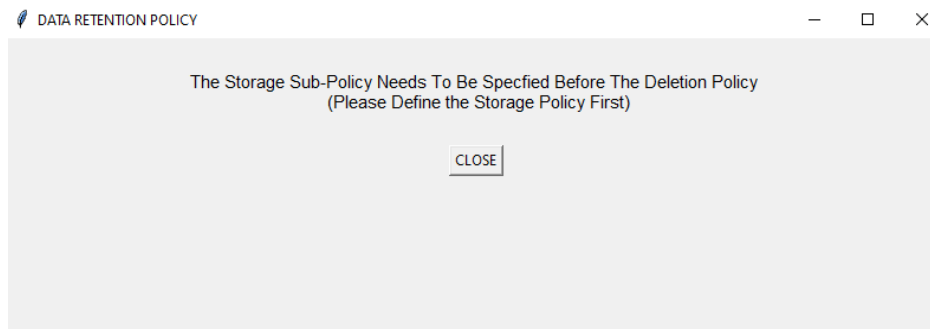
ADD TO WHOM  
(the selected entity will appear in the textbox above)

SAVE TRANSFER POLICY & CLOSE

Figure 28. The specification of the data transfer sub-policy.

### *The data retention sub-policy*

The data retention sub-policy defines when a data type/group will be deleted and from where. In order to define the data retention policy, the users have to define the storage policy first, as the deletion will be defined in the storage places specified in the storage policy. A reminder will be shown to the user if the storage policy has not been specified before the retention policy.



*Figure 29. A data storage sub-policy is closely related to the deletion sub-policy.*

DATA RETENTION POLICY

CHOOSE A DELETION OPTION BELOW (FROM WHERE THE DATA << personalinfo >> WILL BE DELETED BY << sp (service provider) >>)

From Main & Backup Storage

THE RETENTION DELAY OF THE DATA << personalinfo >> IN THE MAIN STORAGE OF << sp (service provider) >>  
(e.g., 2y, 2mo, 2w, 2d, 2h, 2m, 2y+2mo - for 2 years, 2 months, 2 weeks, 2 days, 2 hours, 2 mins)

10y+6mo

THE RETENTION DELAY OF THE DATA << personalinfo >> IN THE BACKUP STORAGE OF << sp (service provider) >>  
(e.g., 2y, 2mo, 2w, 2d, 2h, 2m, 2y+2mo - for 2 years, 2 months, 2 weeks, 2 days, 2 hours, 2 mins)

15y

SAVE DELETION POLICY & CLOSE

Figure 30. A data storage sub-policy is closely related to the deletion sub-policy.

Depending on which storage option was selected in the storage policy the user will be able to select between main or back up storage places. Afterwards, the retention delay can be defined using *y* for year, *mo* for month, *w* for week, *d* for day, *h* for hour and *m* for minute, and their combinations. In Figure 30, for example, the personal information will be kept in the main storage places of the service provider until 10 years and 6 months, and 15 years in the back up storage places.

### The data possession sub-policy

The data possession sub-policy defines who can have/possess a piece of data of a given group. The users only need to specify who are allowed to have or possess a given data group, DataProVe will automatically assume that the rest entities/components are not allowed to have/possess the selected type of data.

DATA POSSESSION POLICY

Is auth (third party authority) allowed to have/possess a data group energy ? (Y if Yes)

Y

SAVE DATA POSSESSION POLICY & CLOSE

Figure 31. The data possession sub-policy.

### *The data connection permitted sub-policy*

This sub-policy specifies which entity is permitted to connect or link two types/groups of data.

DATA CONNECTION/LINKING POLICY - PERMIT POLICY

Choose a (data) group that sp (service provider) is PERMITTED to be able to link with the data of type/group energy

personalinfo

Do you PERMIT sp (service provider) to uniquely link these two types of data ?

No

energy-personalinfo:Only Not Unique Link is Allowed

ADD DATA CONNECTION POLICY

DELETE CONNECTION POLICY

CLOSE & SAVE

Figure 32. The data connection permission sub-policy.

In the second drop-down option menu, the user can specify further if the selected entity is permitted to link pieces of data uniquely, meaning that it will be able to deduce that the two pieces of data belongs to the same individual.

For example, in Figure 32, we specified that the service provider is permitted to be able to link the data group energy and the data group *personalinfo*. However, we do not allow the service provider to be able to uniquely link the two data groups. Obviously, if *personalinfo* was defined as unique, then unique link is always possible, so there is chance that the architecture always violates this requirement of the policy.

### *The data connection forbidden sub-policy*

This sub-policy is the counterpart of the permitted policy. While in case of the data possession policy, the user only needs to specify which entity is allowed to have or possess certain type of data, and DataProVe automatically assumes that the rest are not allowed, here the user needs to explicitly specify which pair of data types/groups are an entity is forbidden to be able to link together.

DATA CONNECTION/LINKING POLICY - FORBID POLICY

Choose a (data) group that sp (service provider) is FORBIDDEN to be able to link with the data of type energy

personalinfo

Do you FORBID sp (service provider) only to uniquely link these two types of data ?

Yes

energy-personalinfo:Unique Link is Forbidden

ADD DATA CONNECTION POLICY

DELETE CONNECTION POLICY

CLOSE & SAVE

Figure 33. The data connection permission sub-policy. The case when only unique link is forbidden.

For example, in Figure 33, we forbid for the third-party authority to be able to link the data group personalinfo with the data group energy. Here, we forbid the unique link of the two data groups for the third-party authority.

If we choose “No”, then it means that any ability to link any two pieces of data of the given data groups, is forbidden (not just unique link). Hence, this option is stricter than the previously one.

DATA CONNECTION/LINKING POLICY - FORBID POLICY

Choose a (data) group that sp (service provider) is FORBIDDEN to be able to link with the data of type energy

personalinfo

Do you FORBID sp (service provider) only to uniquely link these two types of data ?

No

energy-personalinfo:Any Link is Forbidden

ADD DATA CONNECTION POLICY

DELETE CONNECTION POLICY

CLOSE & SAVE

Figure 34. The data connection permission sub-policy.

## Conformance verification:

To verify the conformance of a specified architecture and specified policy, we have click on the button called “VERIFY THE CONFORMANCE OF THE ARCH. AND THE POLICY” at the bottom right corner of the architecture specification page.



Figure 35. The conformance verification of the arch and the policy.

We define three types of conformance, namely, *functional conformance*, *privacy conformance* and the so-called *DPR conformance*.

### *Functional conformance*

The functional conformance captures if an architecture is functionally conforming with the specified policy. Namely:

1. If in the policy, we allow for an entity to be able to have a piece of data of certain data type/group, then in the architecture the same entity can have a piece of data of the same type/group.
2. If in the policy, we allow for an entity to be able to link/uniquely link two pieces of data of certain types/groups, then in the architecture the same entity can link/uniquely link two pieces of data of the same types/groups.
3. If in the policy, the (collection, usage, storage, transfer) consent collection is not required for a piece of data of given type/group, then in the architecture there is no consent collection.
4. If in the policy, we define
  - a. a storage option “Main and Backup Storage” for a piece of data of certain type/group, then in the architecture there is a STORE or STOREAT action defined for both *mainstorage* and *backupstorage*, and for the same data type/group;
  - b. a storage option “Only Main Storage”, then in the architecture there is a STORE or STOREAT action defined for only *mainstorage*, and for the same data type/group.
5. If in the policy, we allow a piece of data of certain type/group, *data*, to be transferred to an entity *ent*, then in the architecture there is **RECEIVEAT**(*ent,data,Time(t)*) or **RECEIVE**(*ent,data*).

#### Violation of the functional conformance

1. In the policy, we allow for an entity to be able to have a piece of data of certain data type/group, but in the architecture the same entity cannot have a piece of data of the same type/group.
2. In the policy, we allow for an entity to be able to link/uniquely link two pieces of data of certain types/groups, but in the architecture the same entity cannot link/uniquely link two pieces of data of the same types/groups.
3. In the policy, the (collection, usage, storage, transfer) consent collection is not required for a piece of data of given type/group, but in the architecture there is a consent collection action (**RECEIVEAT**(sp,**Cconsent**(data),**Time**(t)), or **RECEIVEAT**(sp,**Sconsent**(data),**Time**(t)), or **RECEIVEAT**(sp,**Uconsent**(data),**Time**(t)), or **RECEIVEAT**(third,**Fwconsent**(data,third),**Time**(t))).
4. In the policy, we define
  - a. a storage option “Main and Backup Storage” for a piece of data of certain type/group, but in the architecture, there is **STORE** or **STOREAT** action defined for only either mainstorage or backupstorage, or no store action defined at all, for the same data type/group;
  - b. a storage option “Only Main Storage”, but in the architecture there is no **STORE** or **STOREAT** action defined at all, for the same data type/group.
5. In the policy, we allow a piece of data of certain type/group, *data*, to be transferred to an entity *ent*, but in the architecture there is no **RECEIVEAT**(ent,*data*,**Time**(t)) or **RECEIVE**(ent,*data*) defined (i.e., data is not transferred to the entity *ent*).

#### Privacy conformance

The privacy conformance captures if an architecture satisfies the privacy requirements defined in the policy. Namely:

1. If in the policy, we forbid for an entity to be able to have or possess a piece of data of certain type/group, then in the architecture the same entity cannot have or possess a piece of data of the same type/group.
2. If in the policy, we forbid for an entity to be able to link/uniquely link two pieces of data of certain types/groups, then in the architecture the same entity cannot link/uniquely link two pieces of data of the same types/groups.

#### Violation of the privacy conformance

1. In the policy, we forbid for an entity to be able to have or possess a piece of data of certain type/group, but in the architecture the same entity can/is be able to have or possess a piece of data of the same type/group.
2. In the policy, we forbid for an entity to be able to link/uniquely link two pieces of data of certain types/groups, but in the architecture the same entity can/is be able to link/uniquely link two pieces of data of the same types/groups.

#### DPR conformance

The privacy conformance captures if an architecture satisfies the data protection requirements defined in the policy. Namely:

1. If in the policy, the (collection, usage, storage, transfer) consent collection is required for a piece of data of given type/group, then in the architecture there is a collection for the corresponding consent.
2. If in the policy, we define a (collection, usage, storage) purpose *action:data* for a piece of data of certain type/group, then in the architecture there is the action *action* defined on a compound data type *data*.

#### *Violation of the DPR conformance*

1. In the policy, the (collection, usage, storage, transfer) consent collection is required for a piece of data of given type/group, but in the architecture, there is no collection for the corresponding consent.
2. In the policy, we define a (collection, usage, storage) purpose *action:data* for a piece of data of certain type/group, but in the architecture there is not any action *action* defined on a compound data type *data*, or besides *action*, there are also other actions defined in the architecture on *data* that are not allowed in the policy.
3. In the policy, we define
  - a. a storage option “Main and Backup Storage” for a piece of data of certain type/group, but in the architecture there is a STORE or STOREAT action defined for some storage place, different from *mainstorage* and *backupstorage*, for the same data type/group;
  - b. a storage option “Only Main Storage”, but in the architecture there is a STORE or STOREAT action defined for some storage place, different from *mainstorage*, for the same data type/group.
4. In the policy, we define
  - a. a deletion option “From Main and Backup Storage” for a piece of data of a certain data type/group, *data*, but in the architecture there is no action DELETE(**mainstorage**,*data*) or DELETEWITHIN(**mainstorage**,*data*,Time(value)), or DELETE(**backupstorage**,*data*) or DELETEWITHIN(**backupstorage**,*data*,Time(value));
  - b. a deletion option “Only From Main Storage” for a piece of data of a certain data type/group, *data*, but in the architecture there is no action DELETE(**mainstorage**,*data*) or DELETEWITHIN(**mainstorage**,*data*,Time(value)).
5. In the policy, we allow a piece of data of certain type/group, *data*, to be transferred to an entity *ent*, but in the architecture there is also an action RECEIVEAT(*ent1,data,Time(t)*) or RECEIVE(*ent1,data*) defined for some *ent1* to whom we do not allow data transfer in the policy.

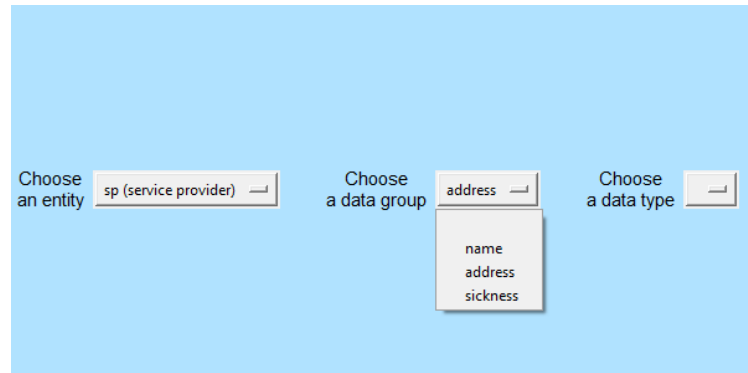


### Application Examples:

In the following, we present the capability of DataProVe through some simple examples.

#### Example 1

In the first example, we specify three data groups in the policy without any data type in them (this is for simplicity purposes.)



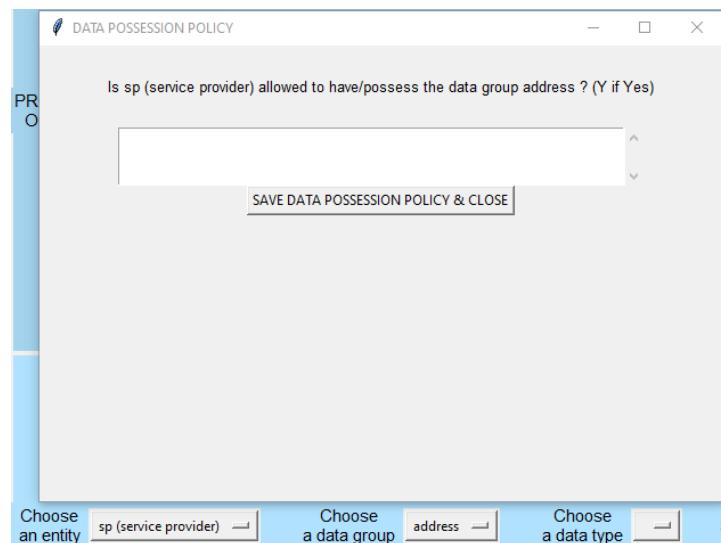
Choose an entity

Choose a data group

Choose a data type

- name
- address
- sickness

Then, we leave the data possession policy blank for all the three data groups above. By default, this means that we do not allow the service provider to be able to have any of these data.



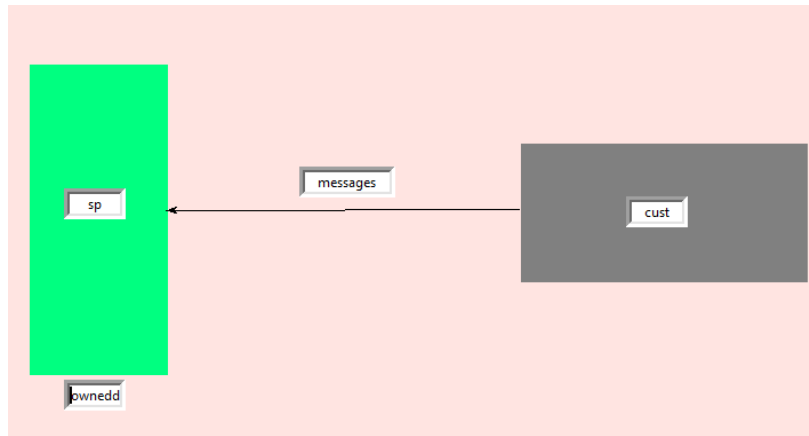
DATA POSSESSION POLICY

Is sp (service provider) allowed to have/possess the data group address ? (Y if Yes)

SAVE DATA POSSESSION POLICY & CLOSE

Choose an entity  Choose a data group  Choose a data type

In the architecture, we define two main components, a green one as the service provider and the grey one represents the customer. The text box *messages* contains the data received by the service provider, while the text box *owneddata* (at the bottom of the green main component) contains the data that sp owns.



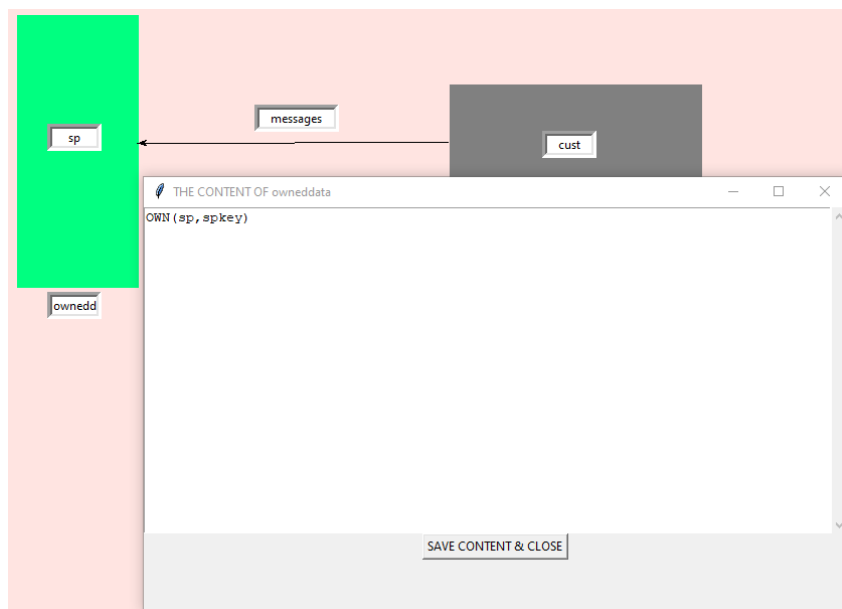
In *messages*, we specify two RECEIVE actions, saying that sp can receive an encrypted account that contains a name and an address inside it. The encryption is symmetric encryption using the shared key spkey.

- **RECEIVE(sp,Senc(Account(name,address),spkey))**
- **RECEIVE(sp,Senc(Nhsrecord(sickness,address),spkey))**

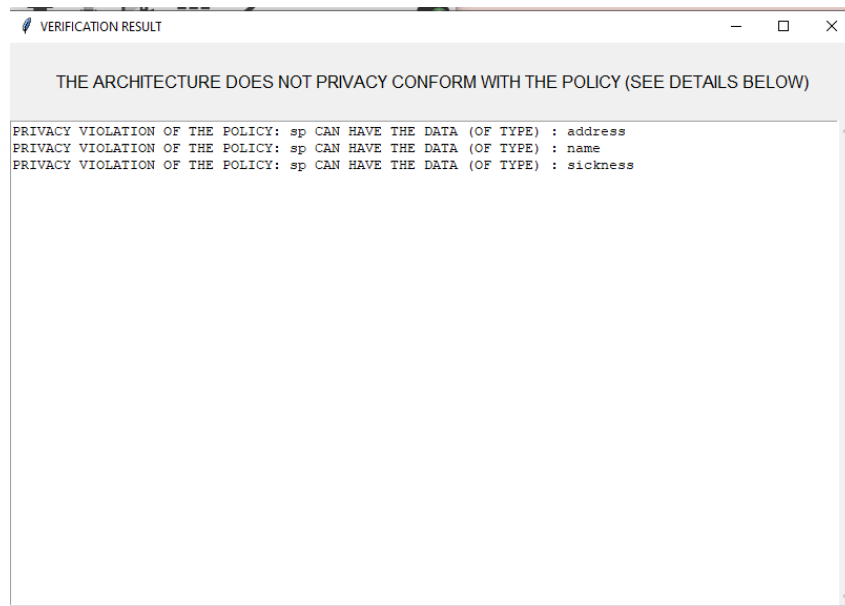


In *owneddata*, we specify an action OWN, saying that sp can own a data of type share key, spkey.

**OWN(sp,spkey)**

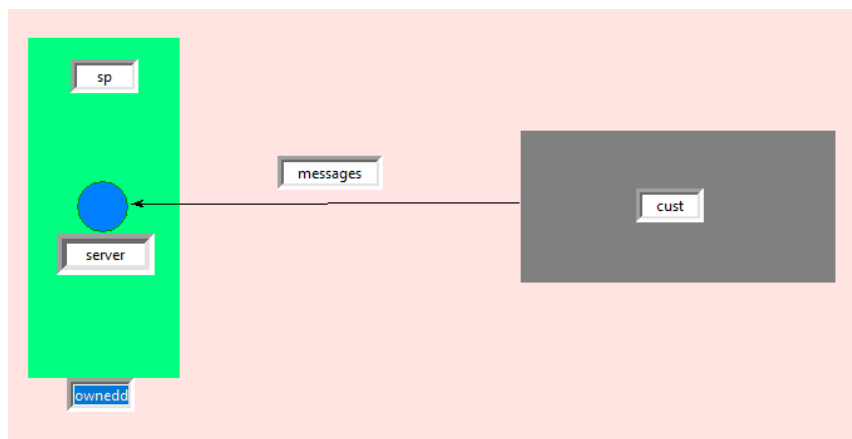


Then, we run the verification, and as result we got privacy violation as the service provider can have both three data groups that we forbid in the policy.



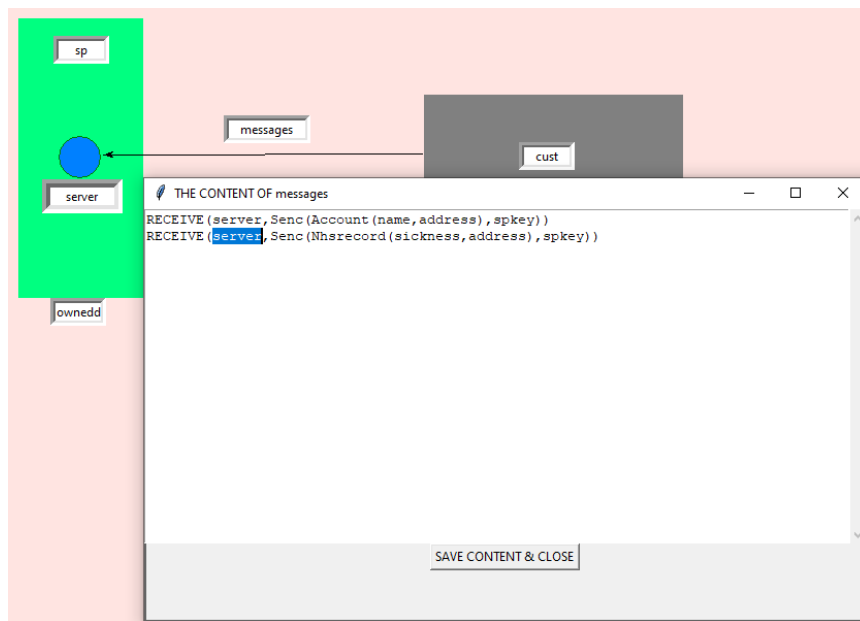
## Example 2 (add sub-component, server)

In the second example, the policy remains the same as the example 1, but in the architecture, we add a sub-component for *sp*, called *server* (blue circle). We located it inside *sp*, to indicate that it is a sub-component of *sp*.



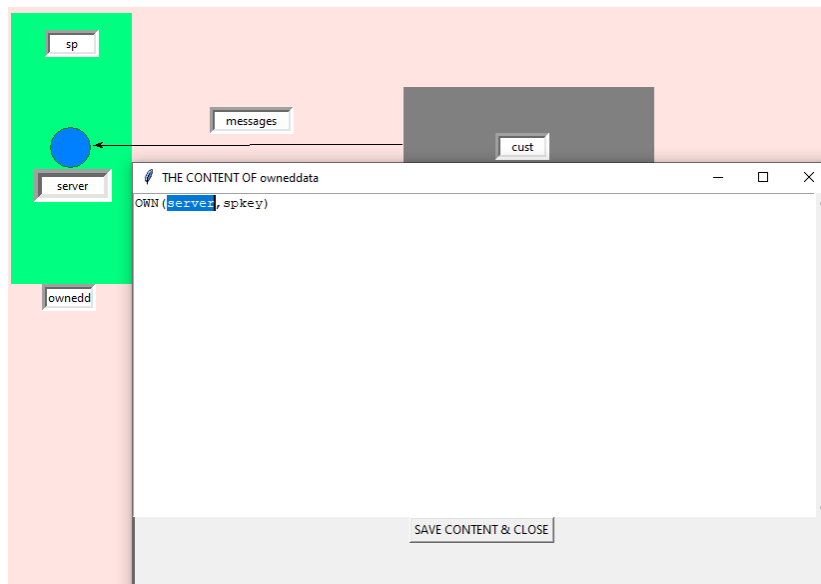
Then, in *messages*, we specify two RECEIVE actions, saying that **server** (instead of *sp*) can receive an encrypted account that contains a name and an address inside it. The encryption is symmetric encryption using the shared key *spkey*.

- **RECEIVE(server,Senc(Account(name,address),spkey))**
- **RECEIVE(server,Senc(Nhsrecord(sickness,address),spkey))**



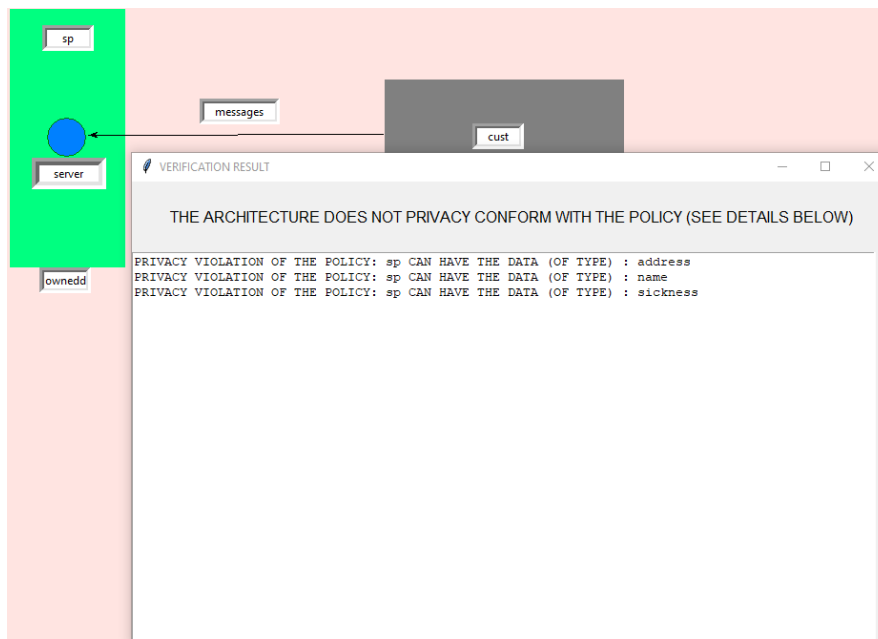
In *owneddata*, we specify an action OWN for server:

**OWN(server,spkey)**



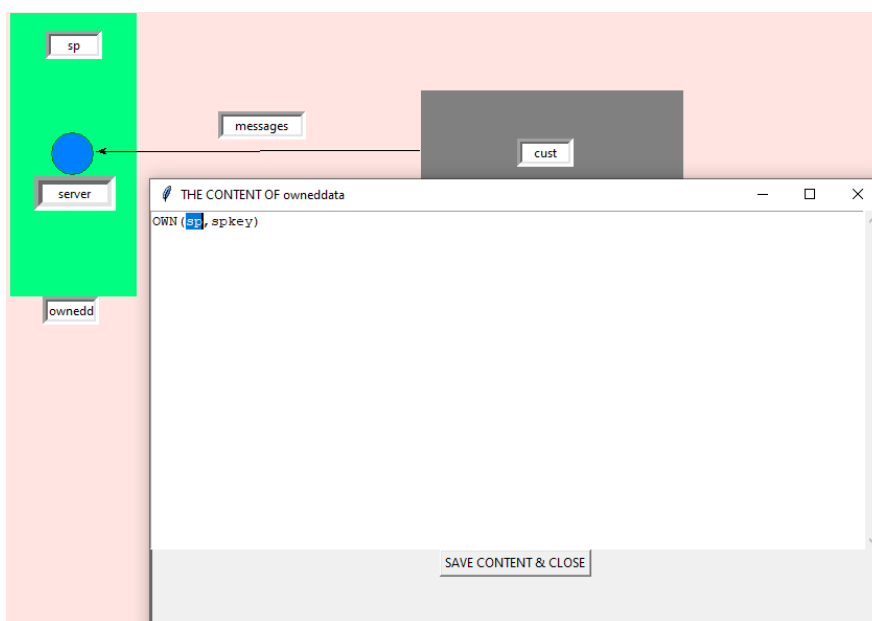
Then, we specified that **sp** can have access to **server** and the data it handles/stores/receives.  
(**sp:server**)

After that, the verification result is the same as in example 1, as the data that server handles is available to sp.

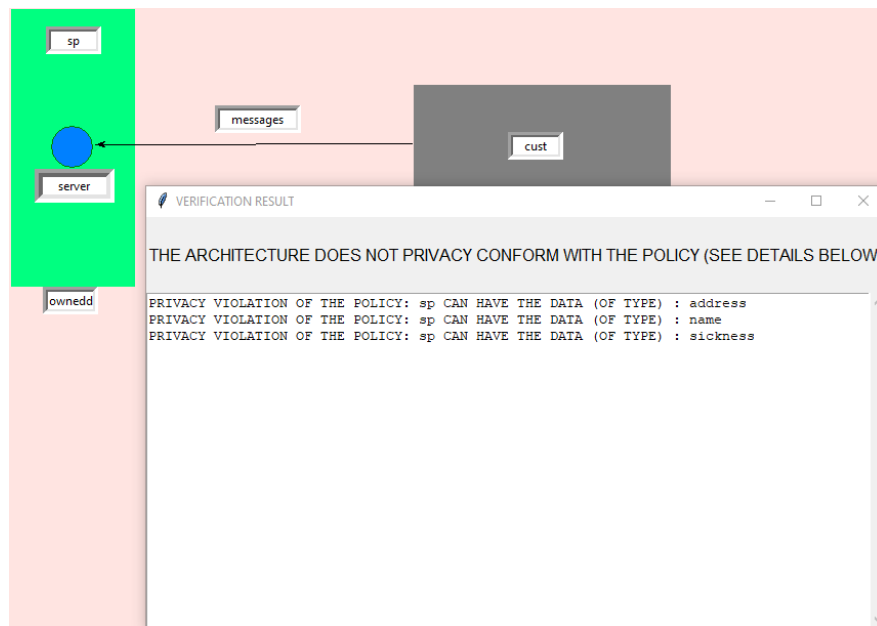


### Example 3

In the third example, the policy remains the same as the previous two examples, but in the architecture, in *messages*, we keep the same RECEIVE messages defined on *server*, but in



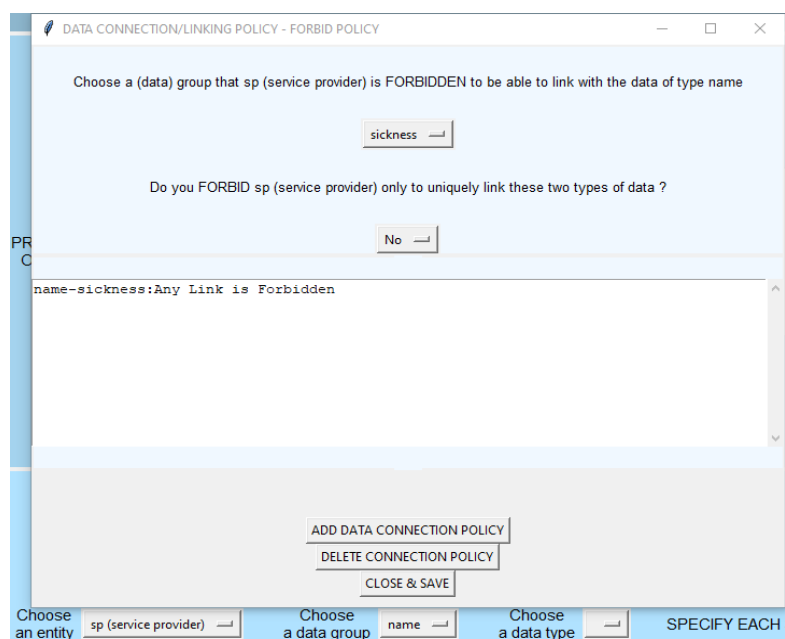
*owneddata*, we specify an action OWN for sp (instead of server). We got the same result after the verification.



This example is only for showing that the user can freely choose among the entities in different messages or actions.

#### Example 4 (Data connection)

Based on the data groups defined in the 1<sup>st</sup> example, in this example, we consider the verification whether an entity is be able to link/connect two pieces of data of type name and sickness.



In the data connection forbid policy, forbid any form of link between a piece of data of type address and sickness for the service provider.

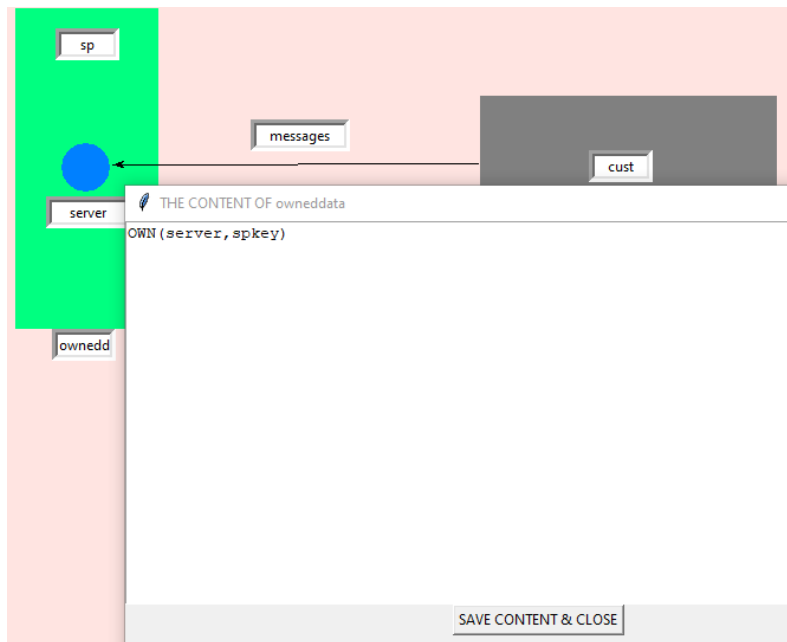
The dialog box is titled "DATA CONNECTION/LINKING POLICY - FORBID POLICY". It contains the following elements:

- A text prompt: "Choose a (data) group that sp (service provider) is FORBIDDEN to be able to link with the data of type address".
- A text input field containing the word "sickness".
- A text prompt: "Do you FORBID sp (service provider) only to uniquely link these two types of data ?".
- A "No" button.
- A text area containing the text: "address-sickness:Any Link is Forbidden".
- Three buttons at the bottom: "ADD DATA CONNECTION POLICY", "DELETE CONNECTION POLICY", and "CLOSE & SAVE".

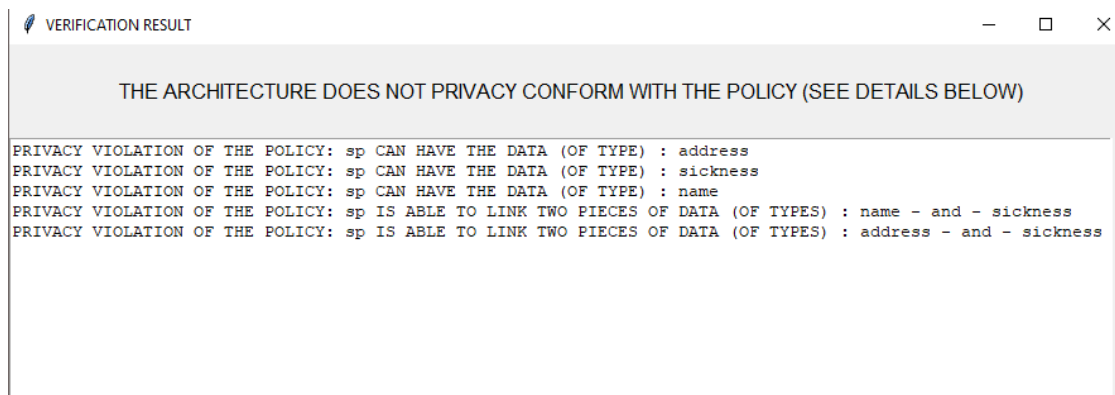
In the architecture, we keep the same components and text boxes and contents of the text boxes. Namely,







Then, we run the verification, and as result we got privacy violation as the service provider can have both three data groups as before, and in addition, it also be able to link name with sickness (see the last line).



### Example 5 (additional sub-component, panel)

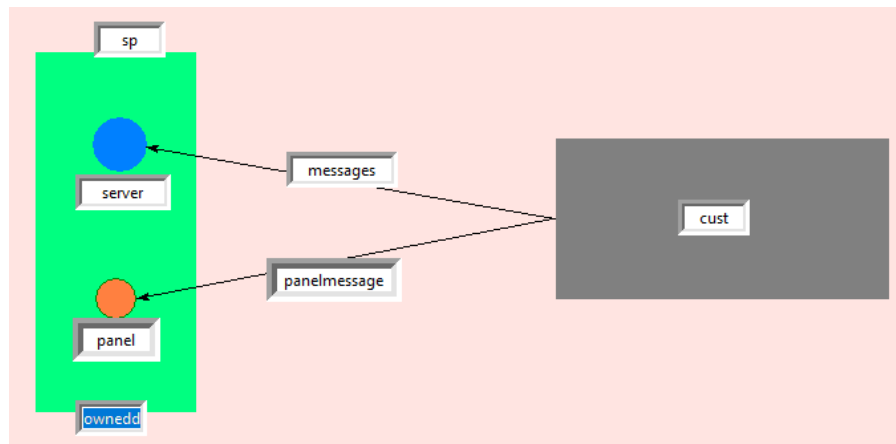
NOW, BESIDES SERVER, WE ADD AN ADDITIONAL SUB-COMPONENT, CALLED PANEL.

Policy:

A service provider cannot have *sickness*, cannot have *name*, and cannot have *address*, as we left the data possession policy blank, which means "No" by default.

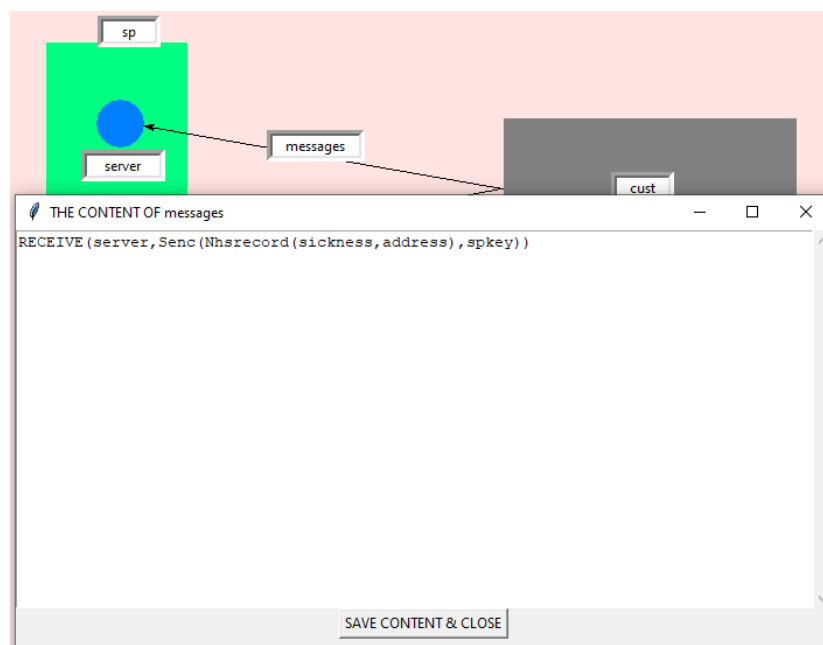
Also, the service provider is not allowed to be able to link *name with sickness* (in any form), and link *address with sickness* (in any form).

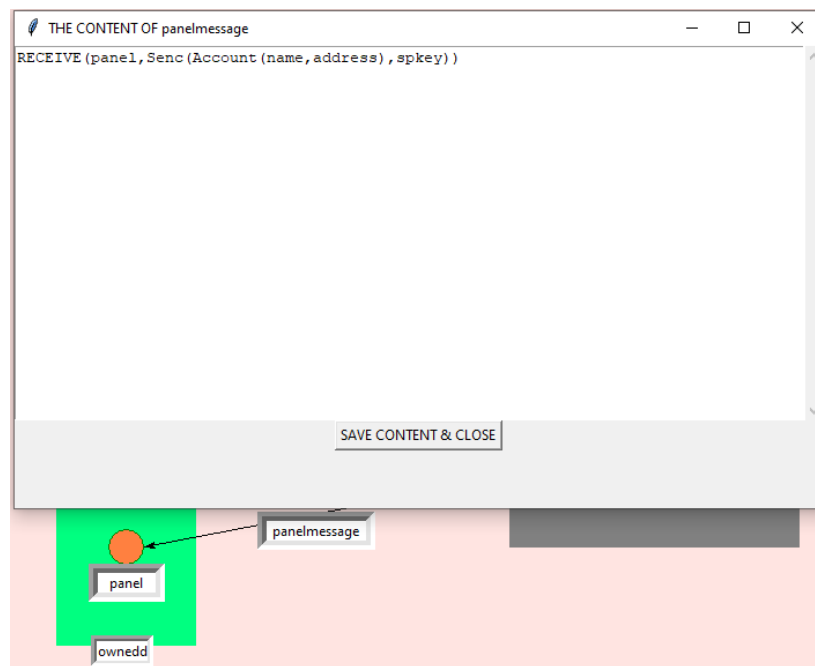
Architecture:



We add the following actions: the server receives a NHS (national health service) record that contains the information about the sickness and address, encrypted with the type of service provider key (shared key), spkey.

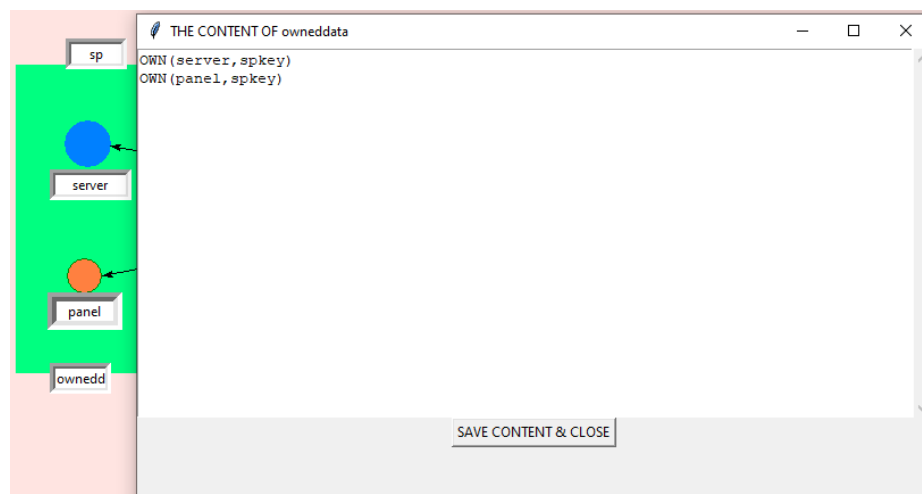
- **RECEIVE**(panel,**Senc**(Account(name,address),spkey))
- **RECEIVE**(server,**Senc**(Nhsrecord(sickness,address),spkey))





We also specify that the panel and server own the key spkey.

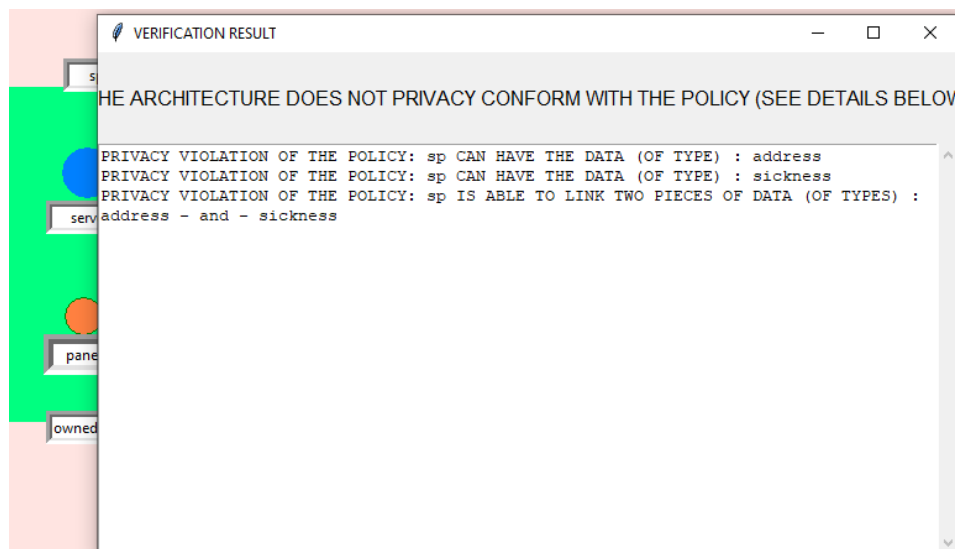
- **OWN(panel,spkey)**
- **OWN(server,spkey)**



We examine two cases:

1st case: we only let the service provider have access to the server but not the data handled or stored by the panel.

Result for the 1st case: Since sp only has access to the server, it can only decrypt the message the server receives with spkey (it has access to any data handled by server) the service provider will get the Nhsrecord in cleartext and the sickness and address info in that. It will also be able to link these two types of info.

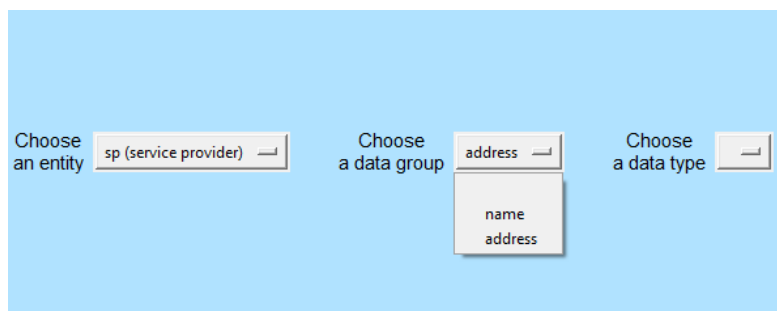


2nd case: We let the service provider to have access to both the server and the panel, and hence, we got the result that sp is able to link two pieces of data of types *name with sickness*, which violates the policy.

### Example 6 (2-layer nested crypto function application)

Policy:

In the policy, we specify two data groups without any data types inside them, for simplicity purposes.



We do not allow the service provider to have a piece of data of type *name*, and *address*, as empty policy field means "NO" (FORBID).

DATA POSSESSION POLICY

Is sp (service provider) allowed to have/possess the data group name ? (Y if Yes)

SAVE DATA POSSESSION POLICY & CLOSE

Choose an entity: sp (service provider) | Choose a data group: name | Choose a data type: SF

DATA POSSESSION POLICY

Is sp (service provider) allowed to have/possess the data group address ? (Y if Yes)

SAVE DATA POSSESSION POLICY & CLOSE

Choose an entity: sp (service provider) | Choose a data group: address | Choose a data type:

#### Architecture:

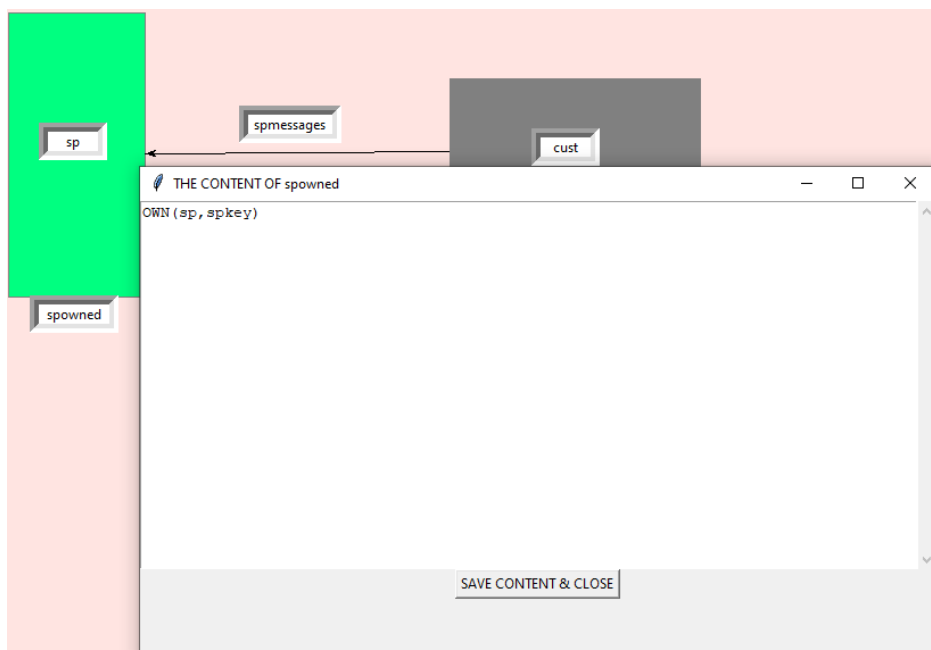
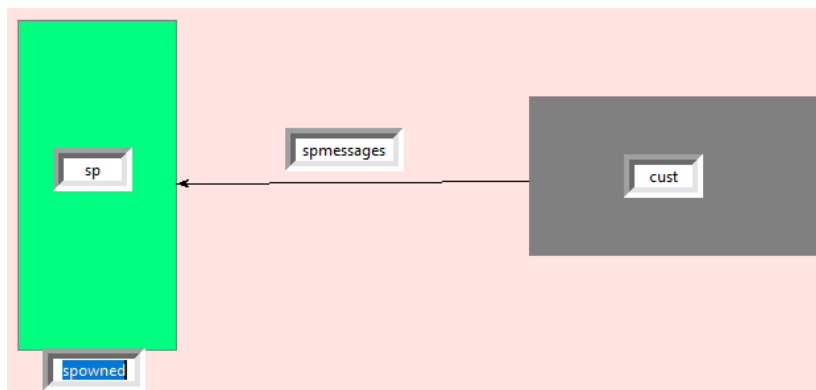
In the architecture we specify with actions that the service provider can receive an account that contains encrypted name (using the shared key spkey), and a cleartext address. The whole account is then again encrypted with the same key, spkey.

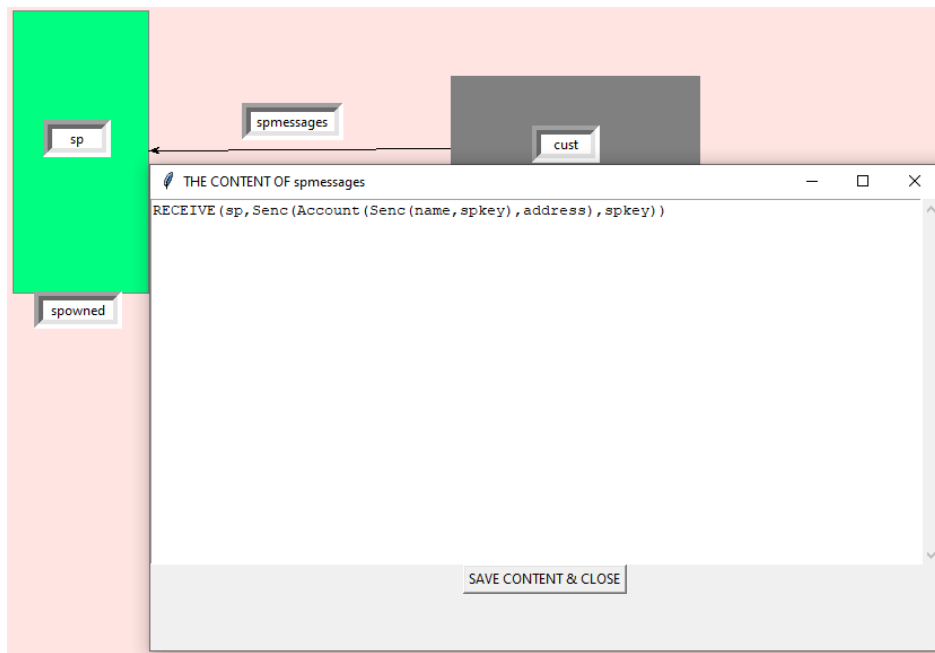
In the architecture level, we abstract away from how the key is shared and what concrete encryption algorithm is used. Those details are left to the protocol or implementation level.

**RECEIVE(sp,Senc(Account(Senc(name,spkey),address),spkey))**

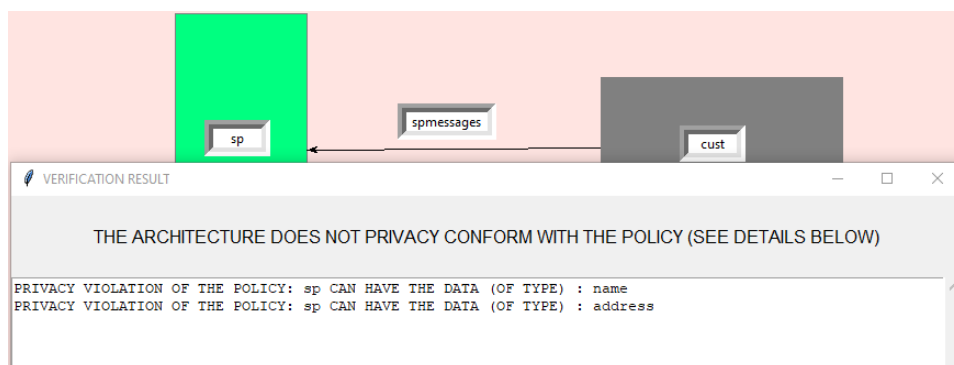
We also specify that the service provider owns spkey.

**OWN(sp,spkey)**





Verification results show privacy violation as sp will be able to use spkey it owns to decrypt the message.



### Example 7 (Asymmetric key encryption)

Here, in the architecture we specify 2-layer nested symmetric key encryption and also apply an asymmetric key encryption.

Policy:

We do not allow the service provider to be able to have a piece of data of type name, and address. Because we left the data possession policy field blank, which means "NO" (FORBID).

Architecture:

The service provider can receive an account that contains encrypted name (using the shared key spkey, Senc is a symmetric key encryption func.), and the cleartext address.

The whole account is then again encrypted with the same key, spkey.



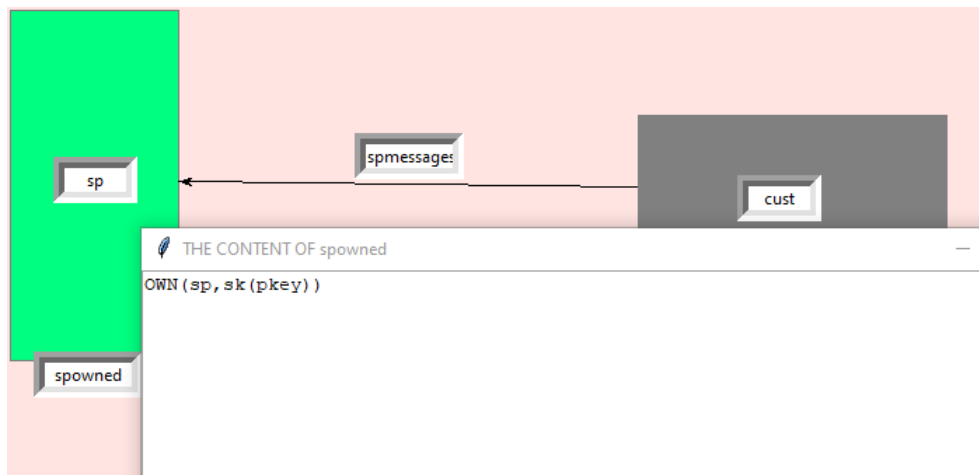
Now, spkey is encrypted using an asymmetric encryption func. (Aenc) with the public key pkey.

Finally, the service provider owns the secret/private key counter part of the public key pkey (namely, sk(pkey)).

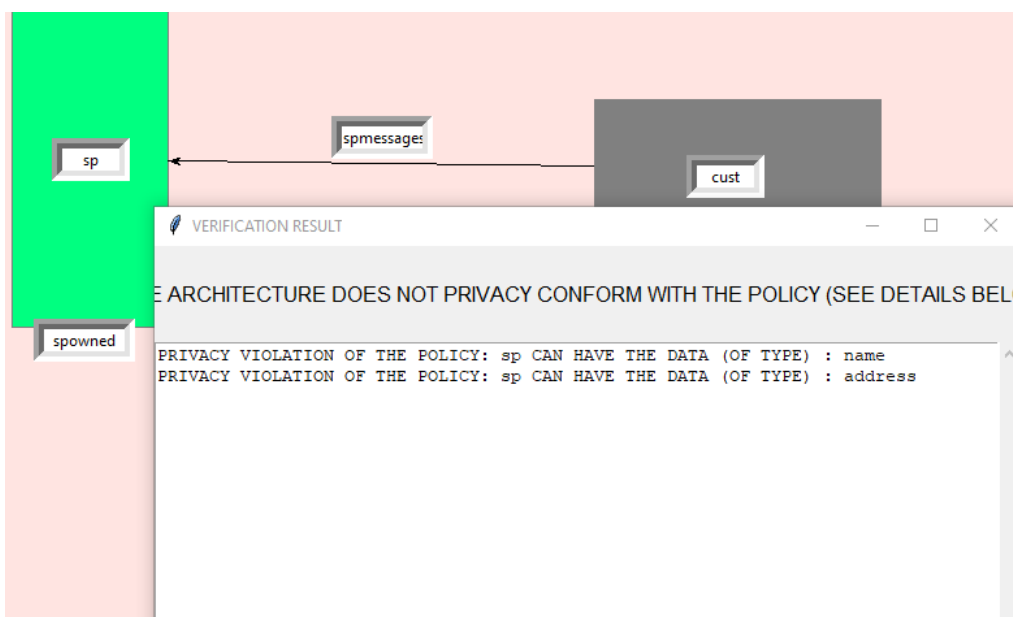
- **RECEIVE(sp,Senc(Account(Senc(name,spkey),address),spkey))**
- **RECEIVE(sp,Aenc(spkey,pkey))**
- **OWN(sp,sk(pkey))**

The content of spawned is **OWN(sp,sk(pkey))**, specifying that sp owns a private key corresponding the public key pkey.





Verification results show privacy violation as sp will be able to use the two keys it owns to decrypt the message.



### Example 8/a (Collection policy, collection consent)

##### DPR COMPLIANCE CASE: #####

Policy:

We require collection consent to be collected at the same time or before a piece of data of type Profile(photo,job) is collected. (here we define compound data type rather than simple, but we could define simple data type as well.)

We also add the collection purpose is to create an account. (This will be used in the example 8/c.)

Architecture:

The service provider receives a profile that contains two pieces of data of type photo and job at some non-specific time t.

**RECEIVEAT(sp,Profile(photo,job),Time(t))**

The service provider receives the corresponding collection consent on the data at the same time or before, this is abstractly modelled by the same time t.

**RECEIVEAT(sp,Cconsent(Profile(photo,job)),Time(t))**



## VERIFICATION RESULT:

We can see that the architecture is DPR comply with the policy. However, we also got the privacy violation because we left the data possession policy blank. By default this means we forbid sp to have *Profile(photo,job)*, but it still can have it in the architecture.



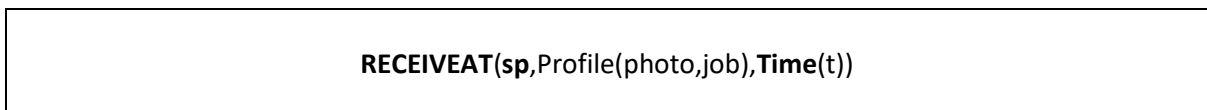
## Example 8/b

##### DPR VIOLATION CASE: #####

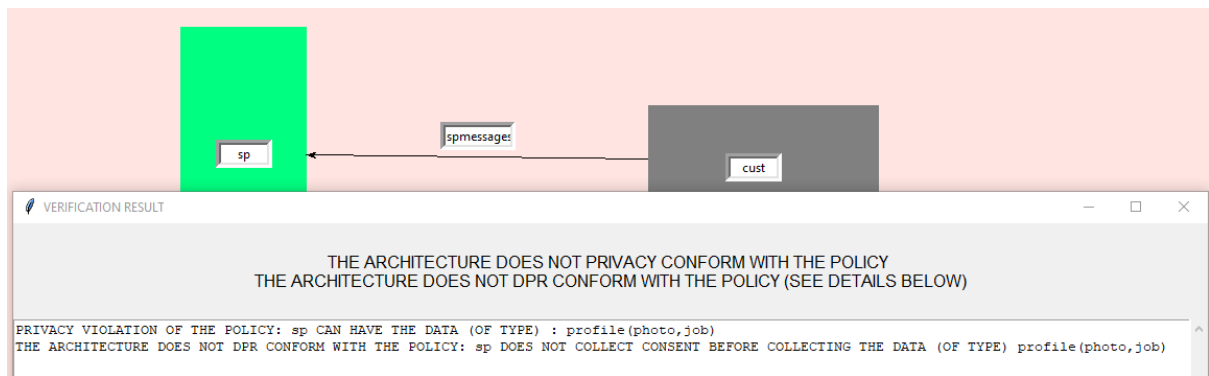
The policy is unchanged compared to the

Architecture:

The service provider receives a profile that contains two pieces of data of type photo and job at some non-specific time t, but no consent collected for that type of data.



Verification result : we can see that the architecture is not DPR comply with the policy.



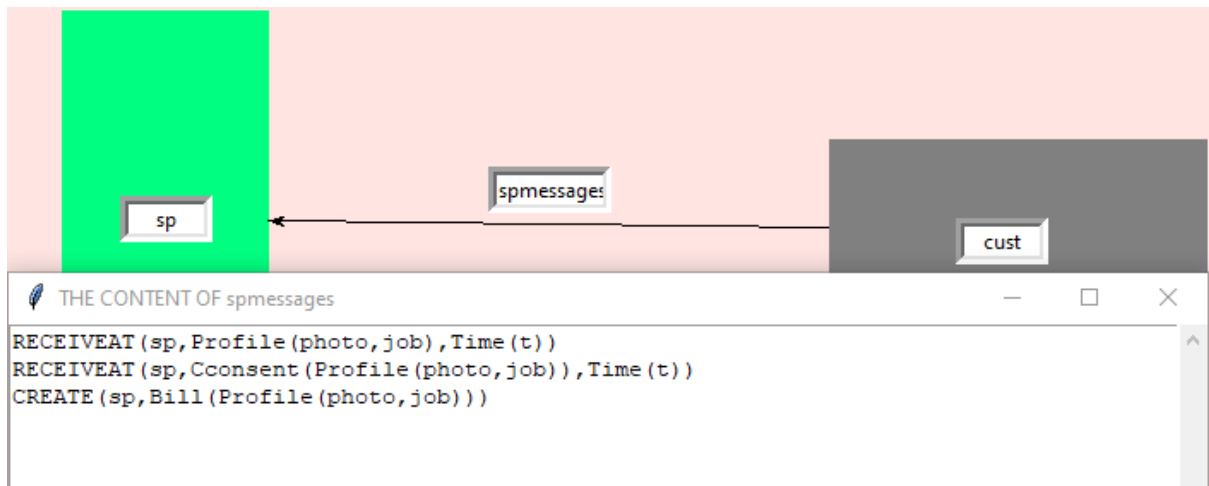
### Example 8/c (Collection purposes)

Policy: Similar to the previous two cases, we require collection consent to be collected at the same time or before a piece of data of type Profile(photo,job) is collected.

We specify the collection purpose as “create an account”.

Architecture: In the architecture, we define an additional action CREATE that says that sp creates a bill based on a profile that contains a photo and a piece of job information.

- **RECEIVEAT**(sp,Profile(photo,job),Time(t))
- **RECEIVEAT**(sp,Cconsent(Profile(photo,job)),Time(t))
- **CREATE**(sp,Bill(Profile(photo,job)))



Verification results:

As we can see, results show that there are DPR violations because (in the 1<sup>st</sup> line) the profile can be collected in the architecture for the purpose of creating a bill, but this is not specified/allowed in the policy. The next lines also DPR violation because we forgot to specify the collection for the data inside the profile, such as photo, and job information.

VERIFICATION RESULT

THE ARCHITECTURE DOES NOT PRIVACY CONFORM WITH THE POLICY  
THE ARCHITECTURE DOES NOT DPR CONFORM WITH THE POLICY (SEE DETAILS BELOW)

THE ARCHITECTURE DOES NOT DPR CONFORM WITH THE POLICY: << sp >> ALSO COLLECTS DATA (OF TYPE) << profile(photo,job) >> IN THE ARCHITECTURE FOR THE FOLLOWING PURPOSES << ('create', 'bill') >> THAT IS NOT DEFINED IN THE POLICY

THE ARCHITECTURE DOES NOT DPR CONFORM WITH THE POLICY: << sp >> ALSO COLLECTS DATA (OF TYPE) << photo >> IN THE ARCHITECTURE, WHICH IS NOT ALLOWED IN THE POLICY

THE ARCHITECTURE DOES NOT DPR CONFORM WITH THE POLICY: << sp >> ALSO COLLECTS DATA (OF TYPE) << job >> IN THE ARCHITECTURE, WHICH IS NOT ALLOWED IN THE POLICY

THE ARCHITECTURE DOES NOT DPR CONFORM WITH THE POLICY: << sp >> ALSO USES DATA (OF TYPE) << profile(photo,job) >> IN THE ARCHITECTURE, WHICH IS NOT ALLOWED IN THE POLICY

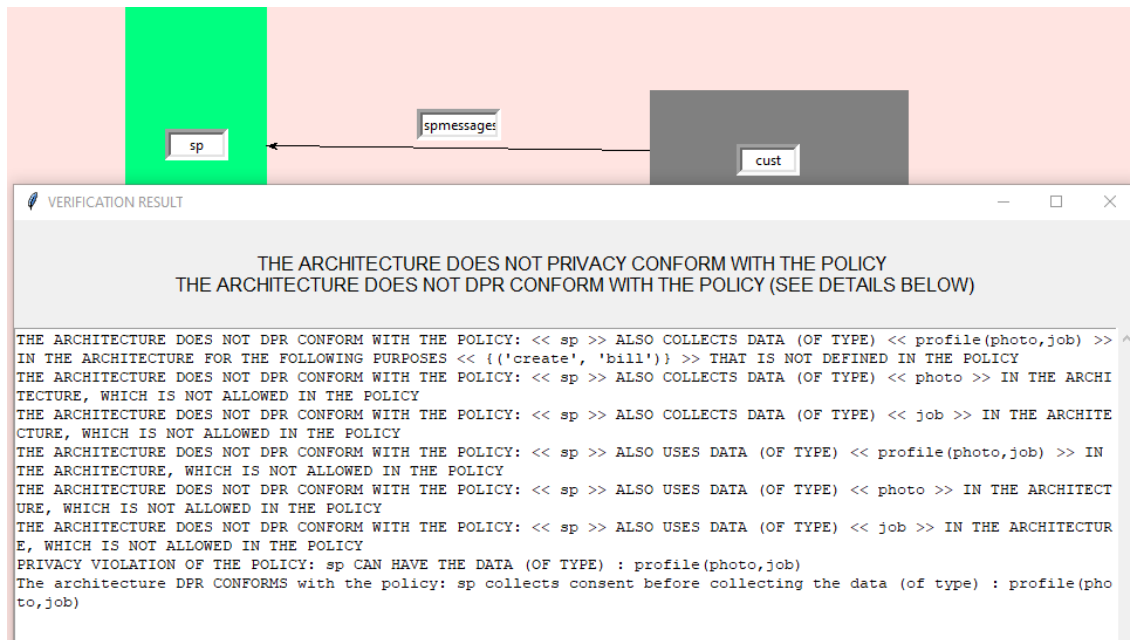
THE ARCHITECTURE DOES NOT DPR CONFORM WITH THE POLICY: << sp >> ALSO USES DATA (OF TYPE) << photo >> IN THE ARCHITECTURE, WHICH IS NOT ALLOWED IN THE POLICY

THE ARCHITECTURE DOES NOT DPR CONFORM WITH THE POLICY: << sp >> ALSO USES DATA (OF TYPE) << job >> IN THE ARCHITECTURE, WHICH IS NOT ALLOWED IN THE POLICY

PRIVACY VIOLATION OF THE POLICY: sp CAN HAVE THE DATA (OF TYPE) : profile(photo,job)

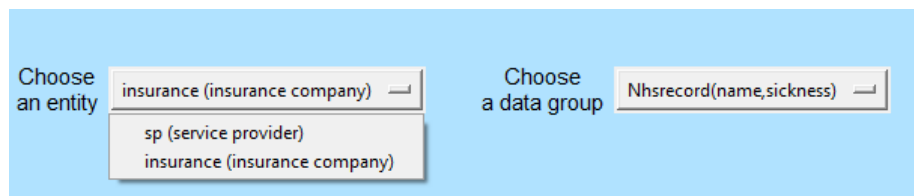
The architecture DPR CONFORMS with the policy: sp collects consent before collecting the data (of type) : profile(photo,job)

Here is the enlarged/zoomed version of the above one:



### Example 9/a (Forward/Transfer policy and consent)

We specify an insurance company besides sp, to whom we set the transfer policy of the data type Nhsrecord that contains a name and sickness information.



In the policy, we require consent for transferring a piece of data of type Nhsrecord (national health service) that contains a name and sickness information to an insurance company (insurance).

DATA TRANSFER POLICY

Does sp (service provider) need to collect consent before transferring the data (of type) Nhsrecord(name,sickness) ? (give Y or N)

Y

Choose to whom << sp (service provider) >> can transfer the data (of type) << Nhsrecord(name,sickness) >>

insurance (insurance company)

insurance

ADD TO WHOM  
(the selected entity will appear in the textbox above)

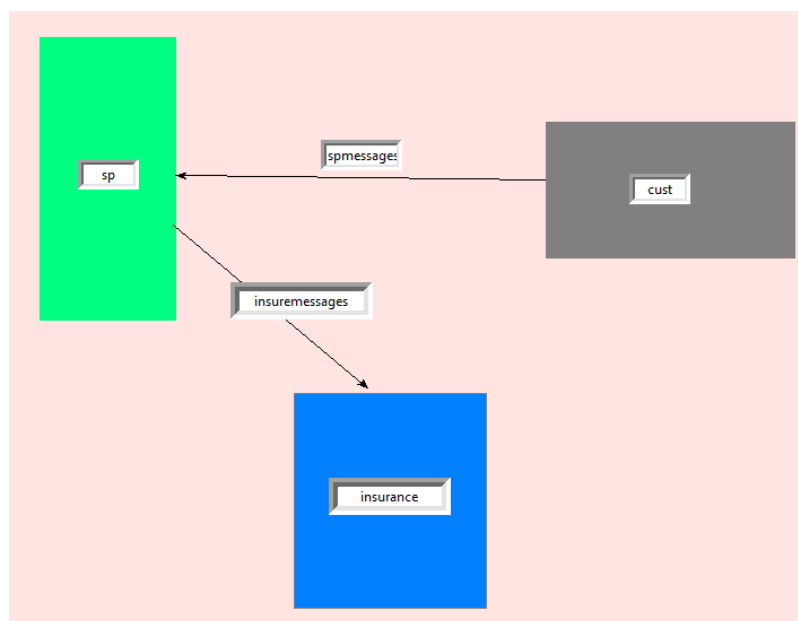
SAVE TRANSFER POLICY & CLOSE

Choose an entity insurance (insurance company) Choose a data group Nhsrecord(name,sickness) Choose a data type SPECIFY E

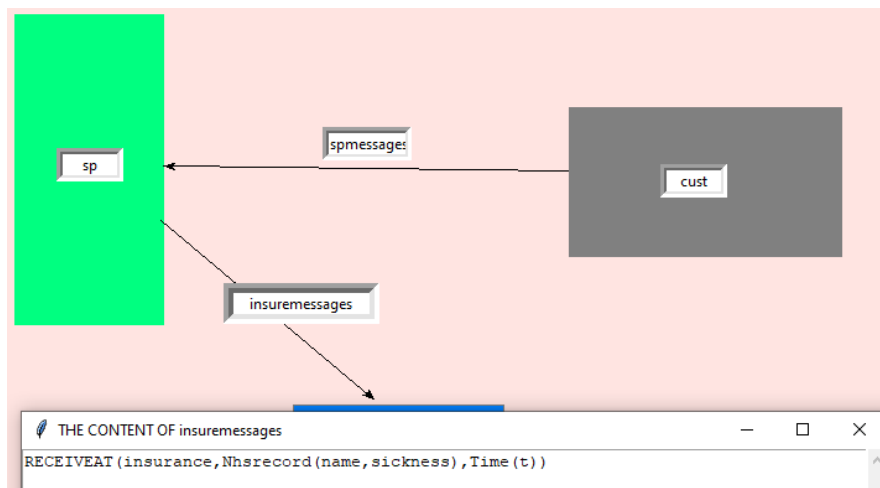
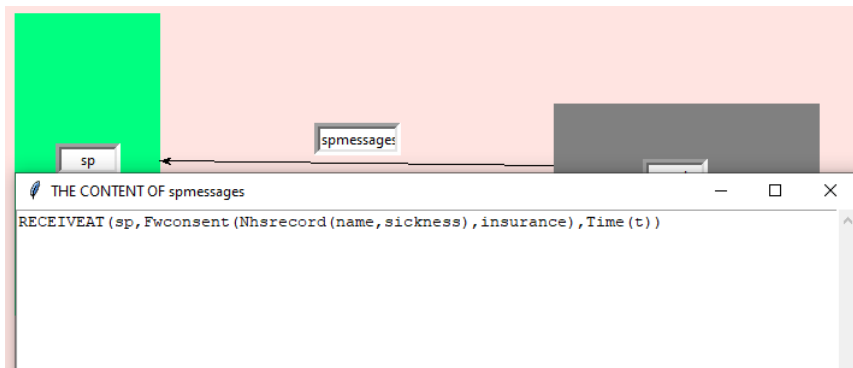
### Architecture:

Whenever the insurance company, insurance, receives a nhsrecord at some non-specific time t, then the service provider must receive consent no later than the time t.

- **RECEIVEAT**(insurance,Nhsrecord(name,sickness),**Time(t)**)
- **RECEIVEAT**(sp,Fwconsent(Nhsrecord(name,sickness),insurance),**Time(t)**)



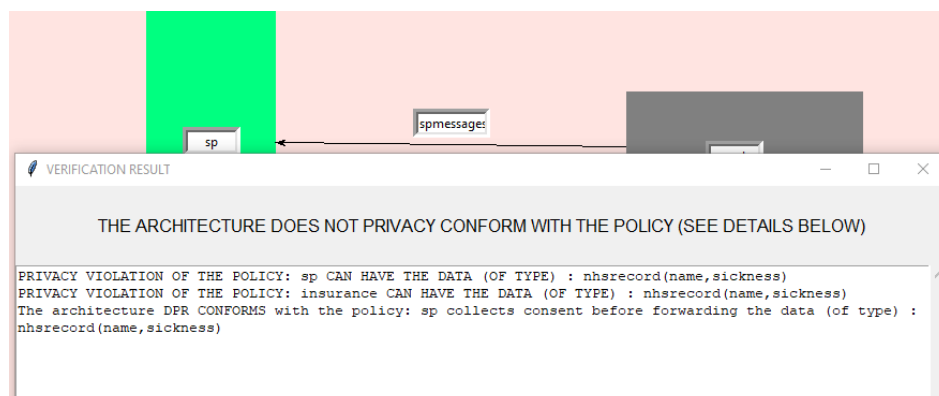
We specify the two messages can be received by sp and insurance:



Verification result:

We got that the architecture is DPR conform with the policy as consent collection was defined.

However, we also get the privacy violation message saying that the service provider and the insurance company can have the data of type NHS record. This is because in the policy we left the data possession policy blank, which by default means that we forbid data possession.





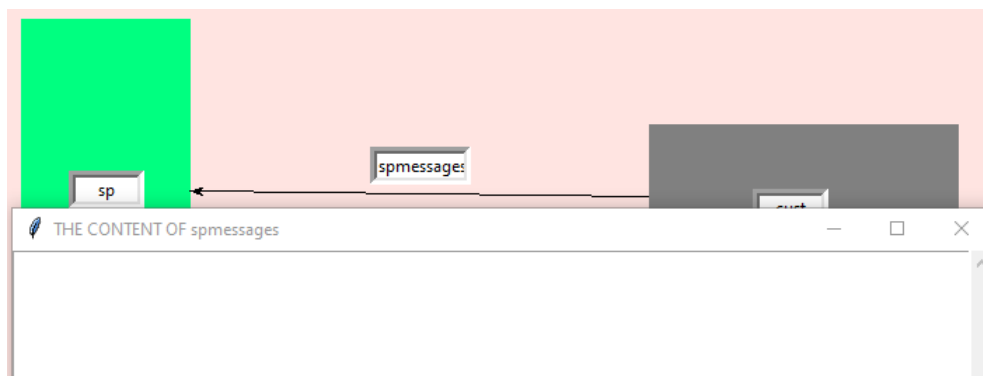
### Example 9/b (===== THE CASE OF DPR VIOLATION =====)

Architecture:

The service provider does not receive any consent, but the insurance company can receive a NHS record. So, we deleted the content of *spmessages* and keep only the message

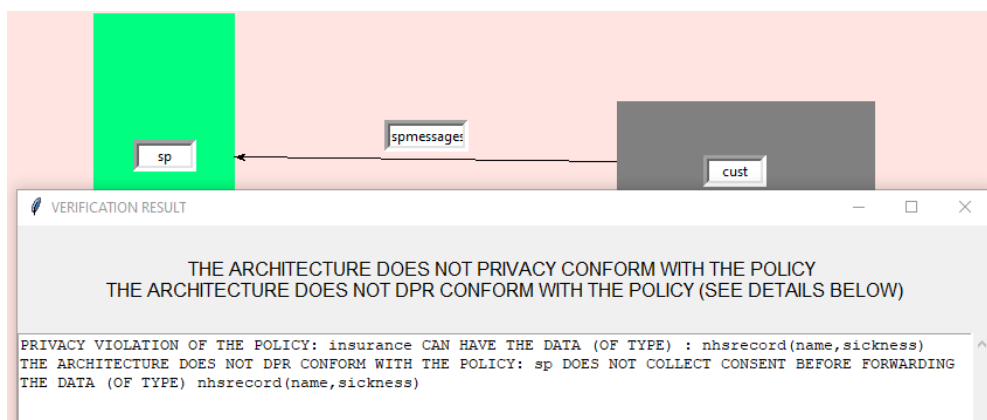
**RECEIVEAT**(insurance,Nhsrecord(name,sickness),**Time**(t))

in *insuremessages*.



Verification result:

We got that the architecture is NOT DPR conform with the policy as consent collection was not defined in the architecture.



## Example 10 (Storage policy/storage consent)

### Policy:

The data of type personalinfo is stored centrally at the service provider, only in the main storage places. Storage consent is set to required.

Again, we leave the data possession policy blank, which forbid for sp to be able to have the data of type personalinfo. We also, leave the collection policy and usage policy blank.

DATA STORAGE POLICY

Does sp (service provider) need to collect consent before the data (of type) personalinfo is stored ? (give Y or N)

Y

Choose a Storage Option (How the Data (of Type) personalinfo Will Be Stored By the Entity sp (service provider) )

Service Provider (Only Main Storage)

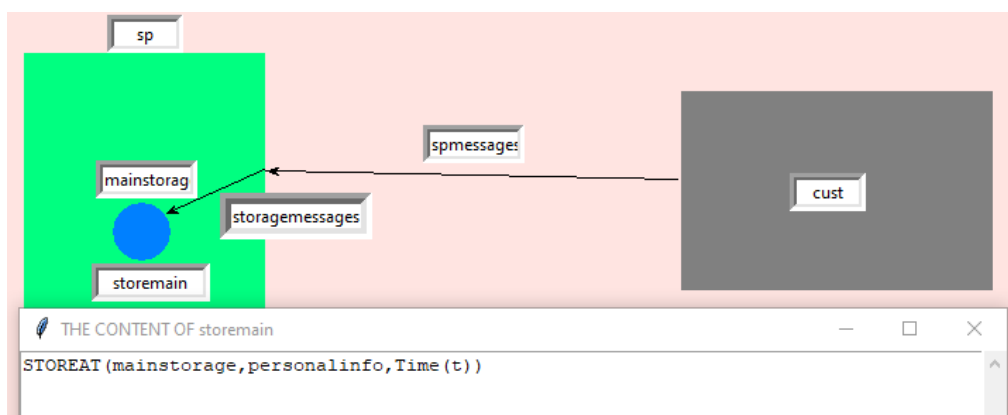
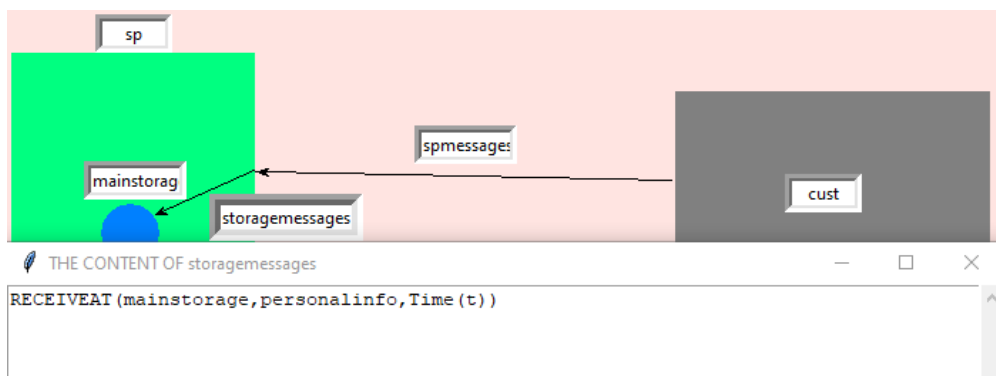
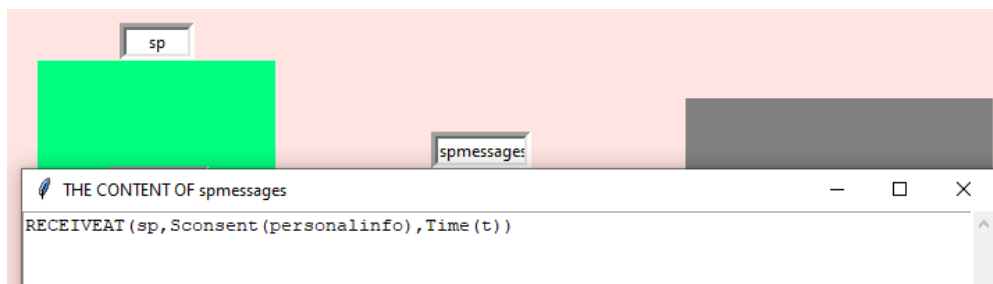
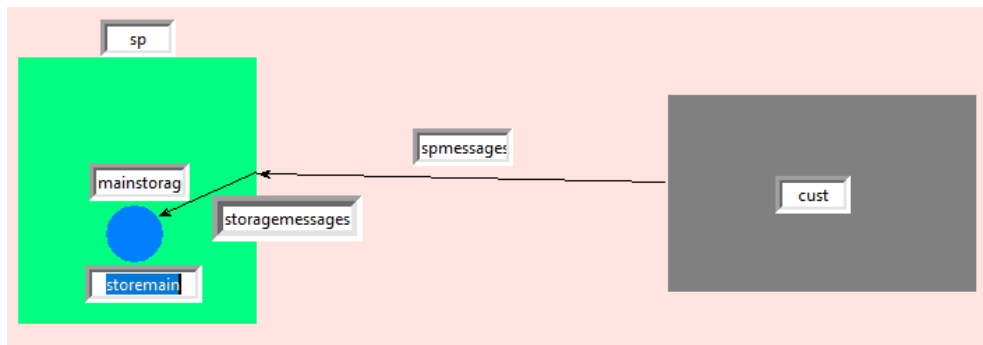
SAVE STORAGE POLICY & CLOSE

Choose an entity sp (service provider) Choose a data group personalinfo Choose a data type SPECIFY EACH SUB-PO

### Architecture:

When a piece of data of type personalinfo is stored in the mainstorage at some non-specific time t, then the service provider (sp) receives the corresponding storage consent no later than time t

- **RECEIVEAT(sp,Sconsent(personalinfo),Time(t))**
- **STOREAT(mainstorage,personalinfo,Time(t))**
- **RECEIVEAT(mainstorage,personalinfo,Time(t))**



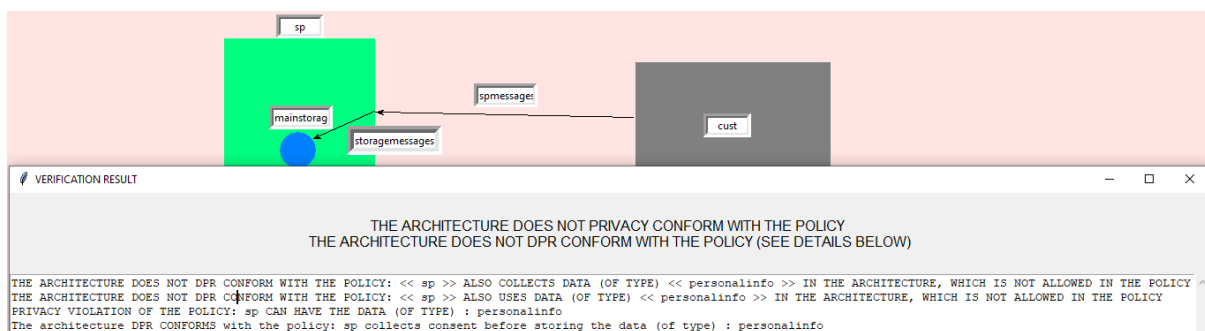
We got the not well-formed architecture message because the architecture is not well-formed, mainstorage store the data of type personalinfo, but has not received it before/or own it/created/calculated it before.

To solve this, we need to add an extra action, when the main storage receives a piece of data of type personalinfo before it can store it (the action can be receive or receiveat).

First of all, we got that the service provider does not have collect consent before storage, which is because we did not specify the relationship between the service provider and mainstorage.

Once we specify that mainstorage is part of sp and sp has access to mainstorage then we will get DPR conformance from sp's point of view.

As you can see after specifying that sp can have access to the data handled by mainstorage, we got DPR conformance (the last line in the figure below).



Similarly to the previous example, if remove the message in spmessage, we will get a DPR violation message.

### Example 11 (Deletion policy)

The deletion policy closely depends on the storage policy. Hence, we build on the previous example about storage policy and keep the storage policy and architecture.

Policy:

Again, we specify one data group *personalinfo* (in this example, we do not set the data types for this group, but we could).

The data of type *personalinfo* is stored centrally at the service provider, only in the main storage places. We also set that storage consent is required.

As for the deletion policy, in the first case, we set the retention delay in the main storage to 8 years (i.e. 8y), while in the second case, to 11 years.

Again, we leave the data possession policy bank, which forbid for sp to be able to have the data of type *personalinfo*.

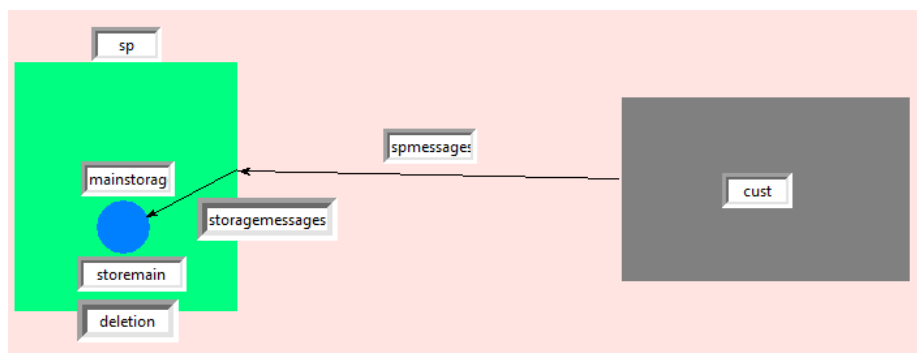
We also, leave the collection policy and usage policy blank.

The screenshot shows a window titled "DATA RETENTION POLICY". At the top, there is a dropdown menu set to "Only From Main Storage". Below this, a text label reads: "THE RETENTION DELAY OF THE DATA << personalinfo >> IN THE MAIN STORAGE OF << sp (service provider) >> (e.g., 2y, 2mo, 2w, 2d, 2h, 2m, 2y+2mo - for 2 years, 2 months, 2 weeks, 2 days, 2 hours, 2 mins)". A text input field below the label contains "8y". At the bottom of the main area is a button labeled "SAVE DELETION POLICY & CLOSE". Below the main area, there are three dropdown menus: "Choose an entity" with "sp (service provider)" selected, "Choose a data group" with "personalinfo" selected, and "Choose a data type" which is empty. To the right of these is a "SPECIFY" button.

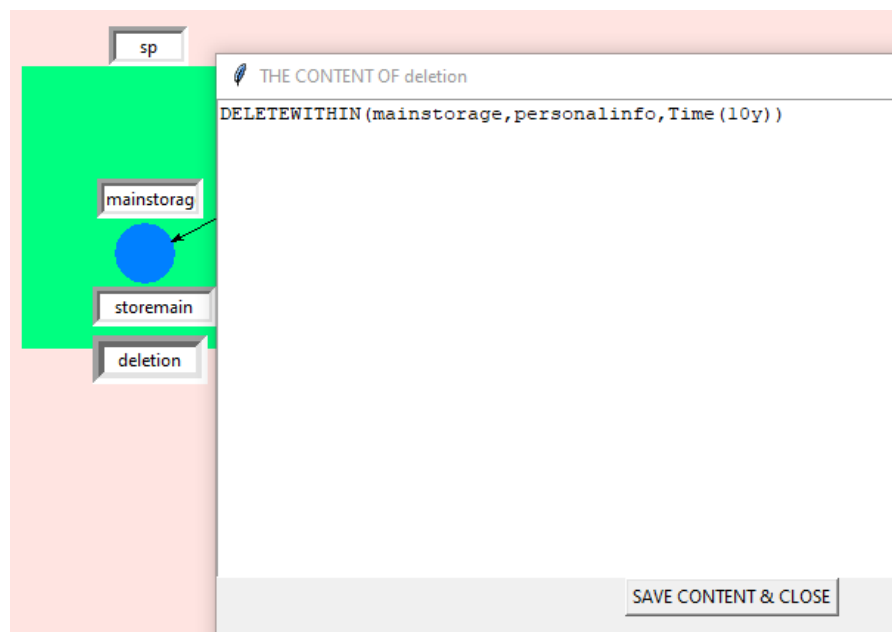
Achitecture:

Besides the same storage and receive actions from the previous storage policy example, we add an action that says a piece of data of type *personalinfo* needs to be deleted from the main storage within 10 years.

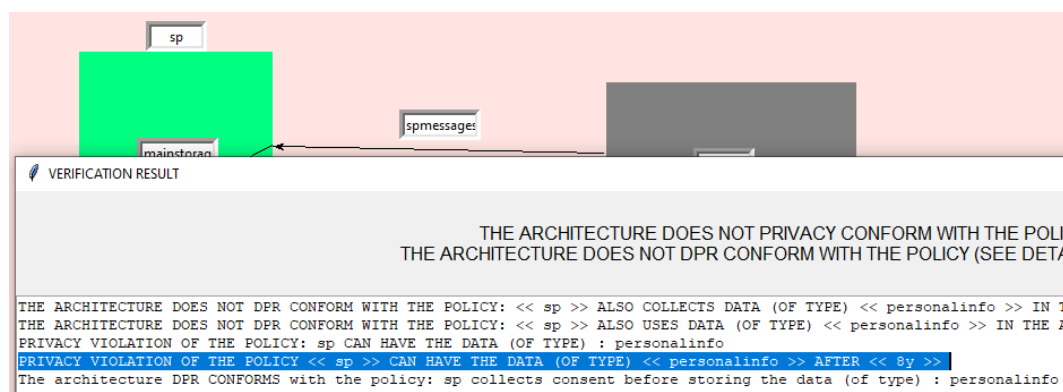
- **RECEIVEAT(sp,Sconsent(personalinfo),Time(t))**
- **STOREAT(mainstorage,personalinfo,Time(t))**
- **RECEIVEAT(mainstorage,personalinfo,Time(t))**
- **DELETEWITHIN(mainstorage,personalinfo,Time(10y)).**



Note that in the DELETEWITHIN action, we provide numerical time value instead of the non-specific time value t.



As a verification result, we got that the architecture violates the privacy property, as the architecture allows for sp to have the data of type *personalinfo* after 8 years, however, in the policy we set it to only 8 years.



### Example 12/a

In this example, we present the receive action with metadata (information about other data) or "packet" header data (IP address, source, destination addresses, etc.).

In DataProVe this kind of information can be specified using the "Meta" construct.

Policy:

In the policy we define four data groups/without data types in them.

Choose an entity

Choose a data group

Choose a data type

nhsnumber  
name  
photo  
address

Then, we forbid (any kind of likability, not only unique link) for the service provider to be able to link two pieces of data of types *nhsnumber* (national health service number), and *photo*.

DATA CONNECTION/LINKING POLICY - FORBID POLICY

Choose a (data) group that sp (service provider) is FORBIDDEN to be able to link with the data of type nhsnumber

Do you FORBID sp (service provider) only to uniquely link these two types of data ?

nhsnumber-photo:Any Link is Forbidden

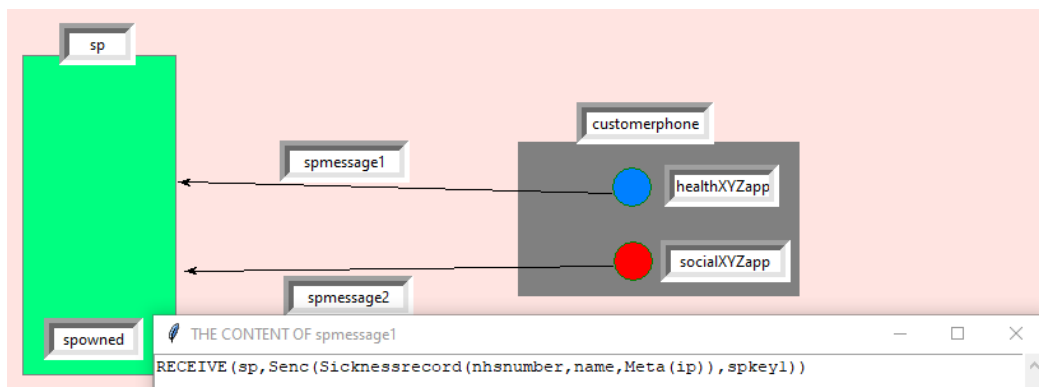
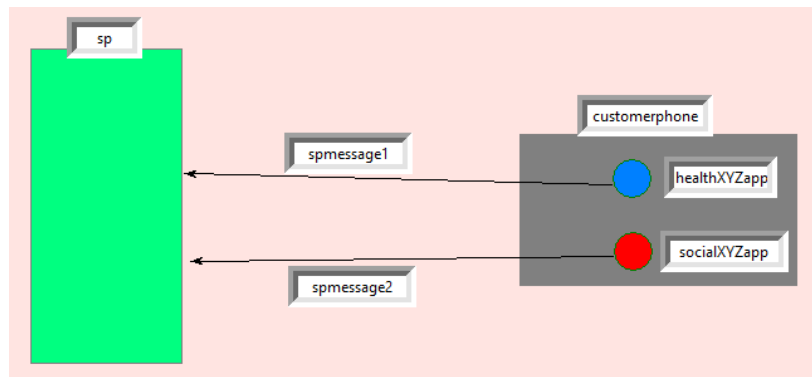
ADD DATA CONNECTION POLICY  
DELETE CONNECTION POLICY  
CLOSE & SAVE

Choose an entity  Choose a data group  Choose a data type  SPECIFY

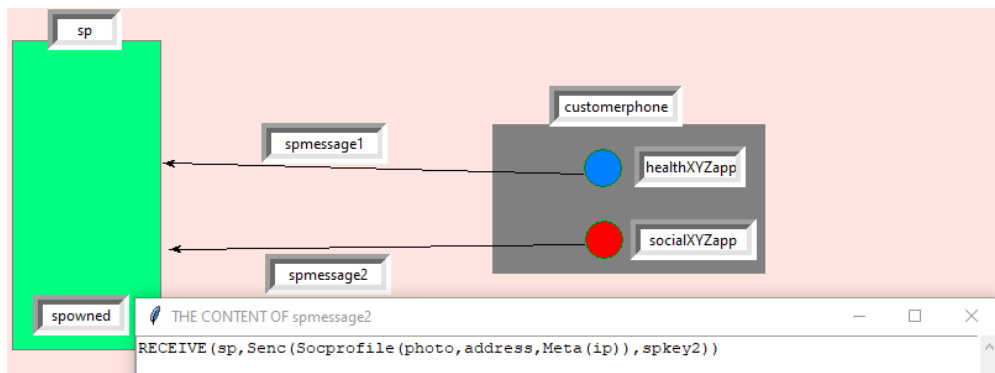
Architecture:

In the architecture, a service provider collects data from two phone applications. The "HealthXYZ" app sends the service provider a sickness record with an ip address (phone ip) other app, called, "SocialXYZ" also sends the social profile with the same ip address (same phone).

- **RECEIVE**(sp,Sicknessrecord(nhsnumber,name,**Meta**(ip)))
- **RECEIVE**(sp,Socprofile(photo,address,**Meta**(ip)))

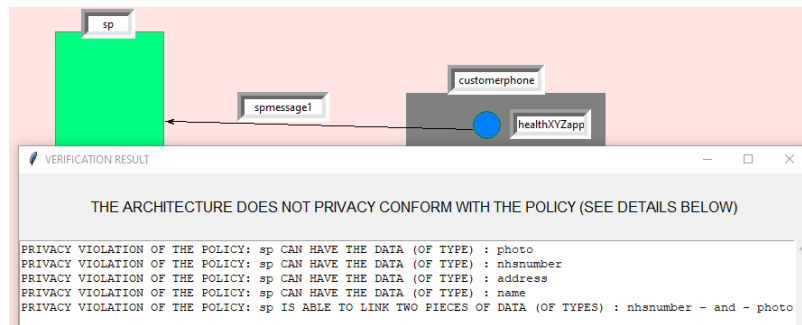


Here we define the two messages *spmessages1* and *spmessages2*.



As a result, we got that the service provider not only be able to link the data of types *nhnumber* with the data of type *photo*, but it also has all the data of types *nhnumber*, *name*, *photo* and *address*.





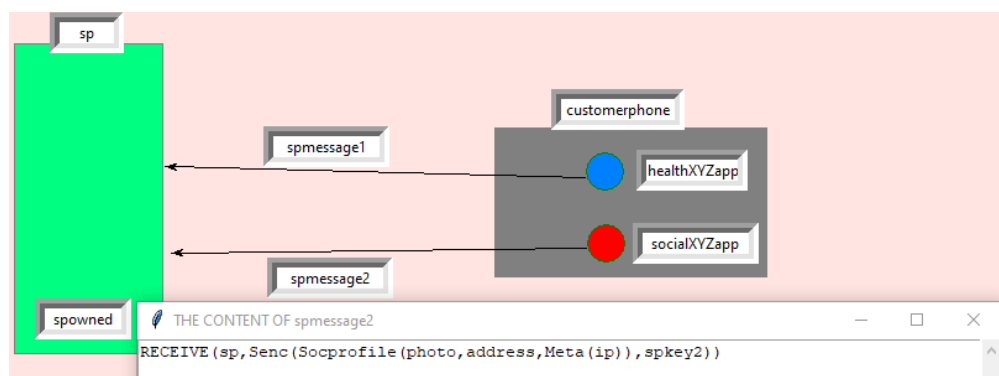
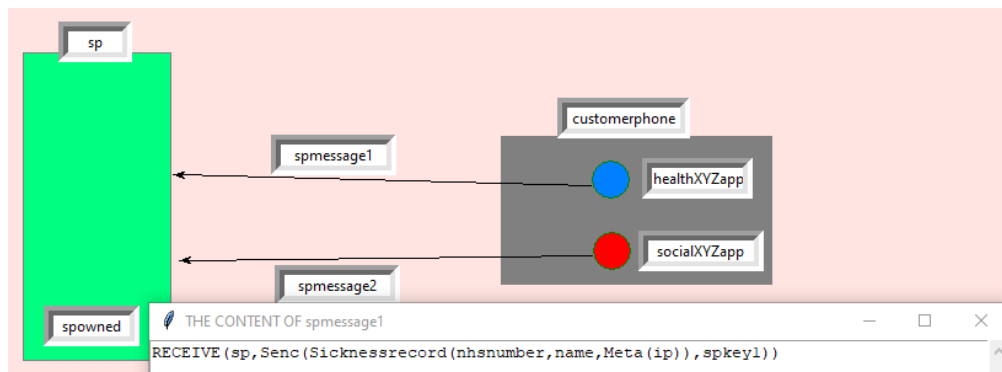
### Example 12/b

Here, the policy is the same as in the point 12/a.

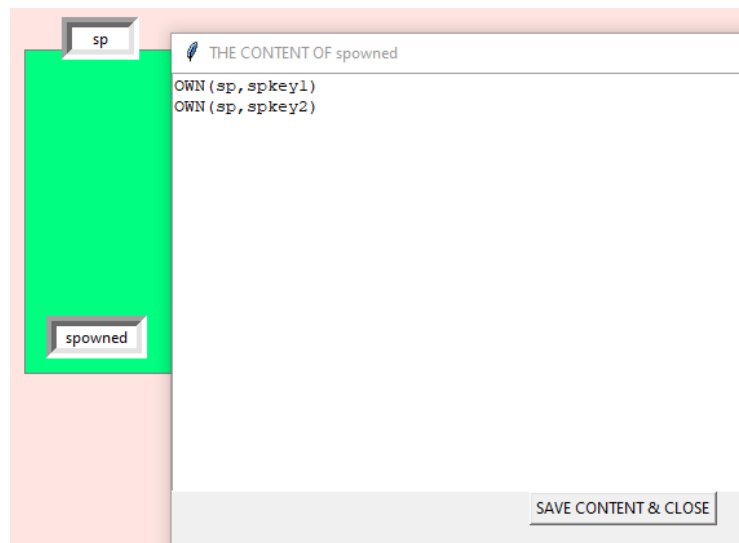
Architecture: The service provider, *sp*, receives the same data but with symmetric encryption, and it owns the decryption key for both messages.

- **RECEIVE**(*sp*, Senc(Sicknessrecord(*nhsnumber*, *name*, **Meta**(*ip*)), *spkey1*))
- **RECEIVE**(*sp*, Senc(Socprofile(*photo*, *address*, **Meta**(*ip*)), *spkey2*))
- **OWN**(*sp*, *spkey1*)
- **OWN**(*sp*, *spkey2*)

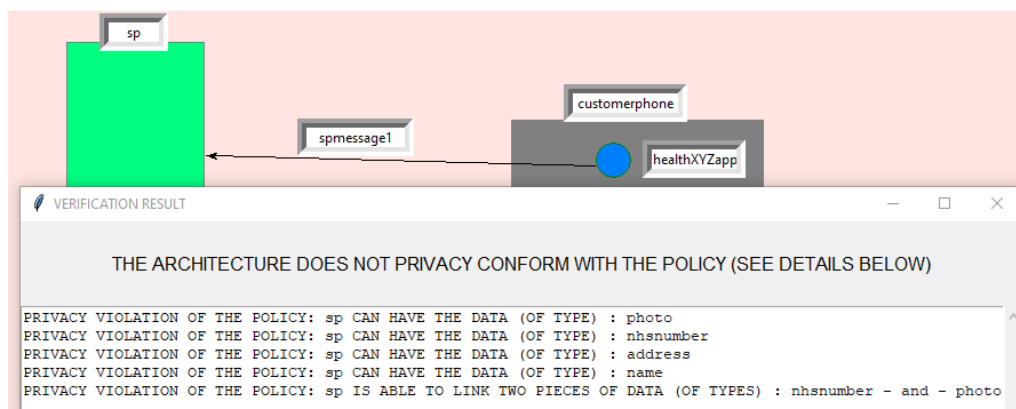
The messages are unchanged compared to the previous example:



Here we also specify the ownership of the two decryption keys.



As you can see, you got the same verification results as the point 12/a, because sp will be able to decrypt both messages and link, have the data inside them.

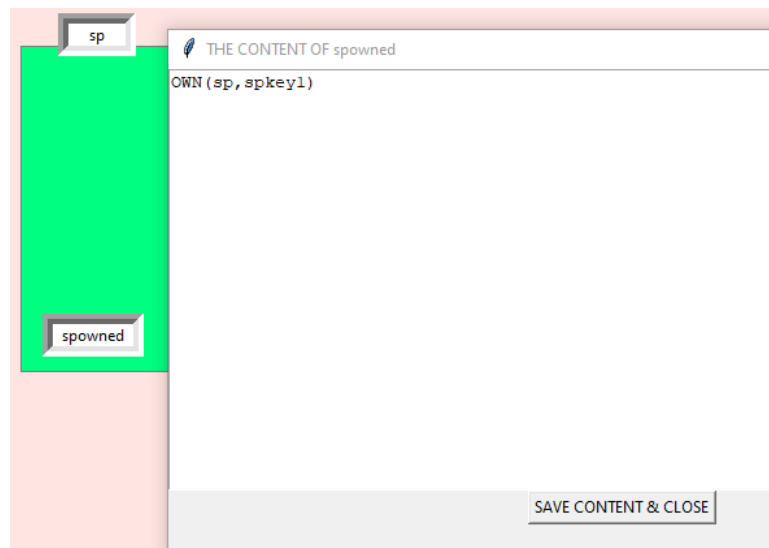


### Example 12/c

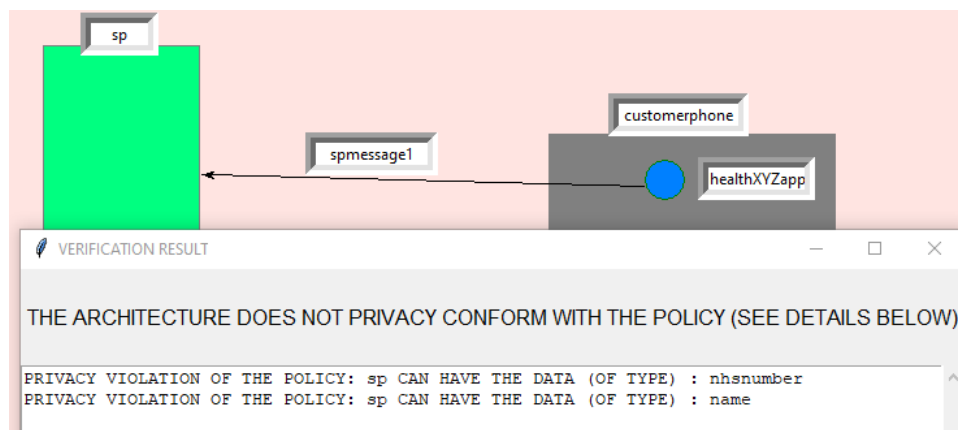
Here, the policy is the same as in the point 12/a.

Architecture: Now sp only owns the decryption key for the first message.

- **RECEIVE**(sp,Senc(Sicknessrecord(nhsnumber,name,**Meta**(ip)),spkey1))
- **RECEIVE**(sp,Senc(Socprofile(photo,address,**Meta**(ip)),spkey2))
- **OWN**(sp,spkey1)



As you can see, the service provide will not be able to link *nhsnumber* with *photo* anymore, but it can still have the data in the first message using the decryption key.



### Example 13 (Pseudonym application)

In the version 0.9, DataProVe uses the keyword **ds** to denote the real identity of the data subject, while **P(ds)** defines the type of the pseudonym of **ds**.

Policy specification:

We specify the policy such that trusted is allowed to be able to have Sickness(disease,ds). In this version, **ds** is a preserved keyword refers to data subject, which captures the real identity of a data subject. The pseudonymised version of this is **P(ds)**, namely, a pseudonym.

DATA POSSESSION POLICY

Is trusted (t) allowed to have/possess the data group Sickness(disease,ds) ? (Y if Yes)

Y

SAVE DATA POSSESSION POLICY & CLOSE

Choose an entity trusted (t) Choose a data group Sickness(disease,ds) Choose a data type

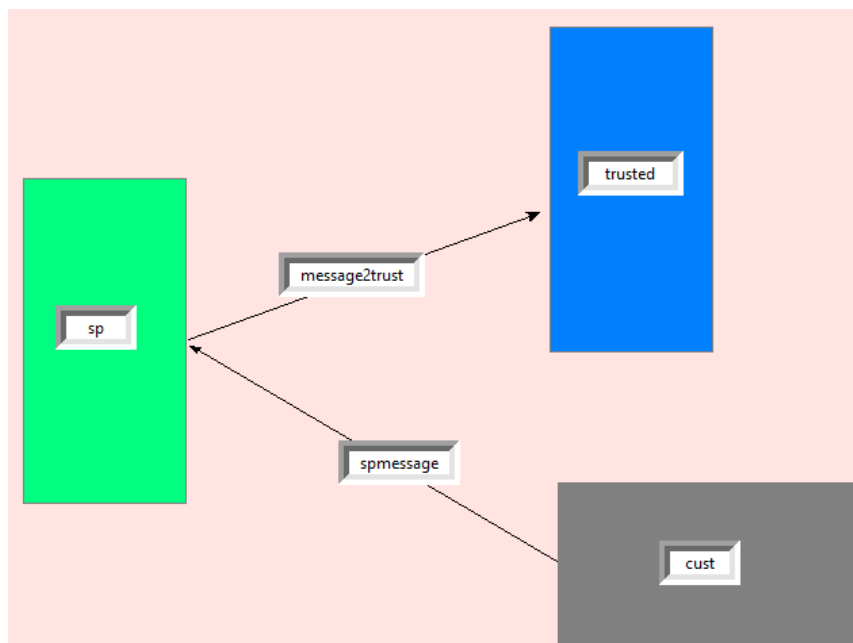
DATA POSSESSION POLICY

Is sp (service provider) allowed to have/possess the data group Sickness(disease,ds) ? (Y if Yes)

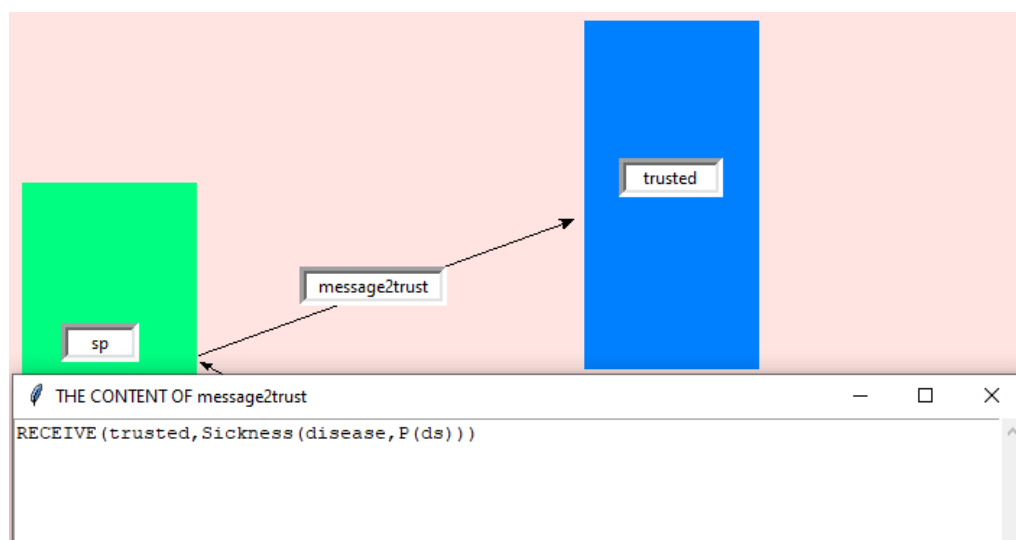
SAVE DATA POSSESSION POLICY & CLOSE

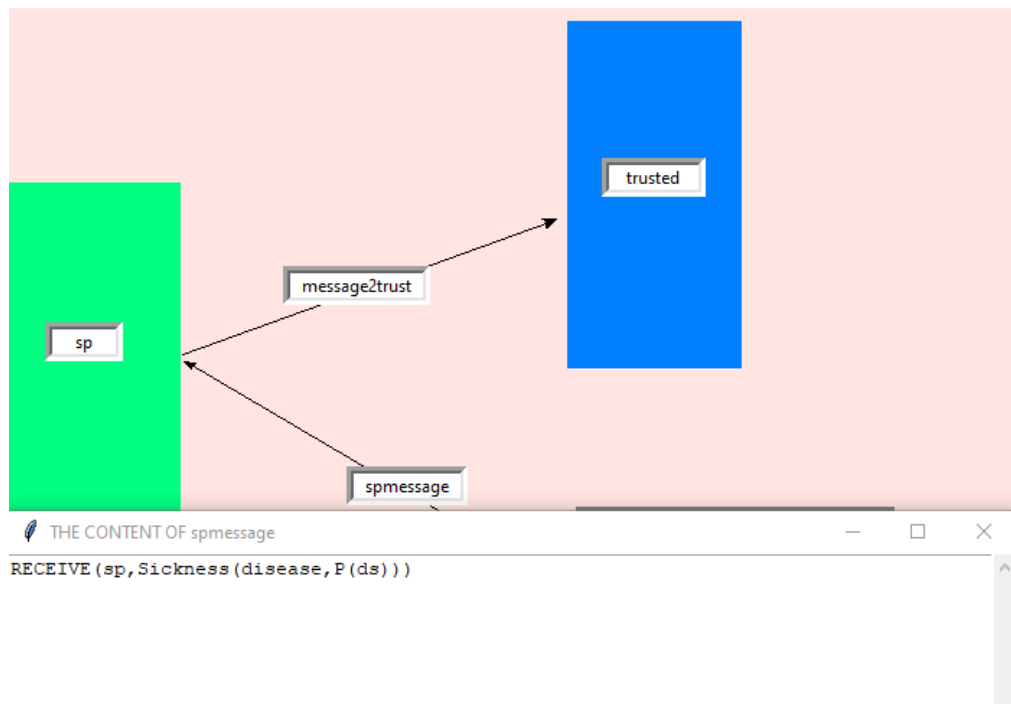
Choose an entity sp (service provider) Choose a data group Sickness(disease,ds) Choose a data type

Architecture specification:



The trusted party can receive a sickness record that contains a piece of information about a disease, and a pseudonym related to the real identity **ds**, **P(ds)**.





### Verification result (conformance):

VERIFICATION RESULT

THE ARCHITECTURE PRIVACY CONFORMS WITH THE POLICY  
THE ARCHITECTURE FUNCTIONALLY CONFORMS WITH THE POLICY  
THE ARCHITECTURE DPR CONFORMS WITH THE POLICY (SEE DETAILS BELOW)

The architecture functionally conform with the policy: trusted can have the data (of type) sickness(disease,ds)

Then, we change the policy such that trusted is not allowed to be able to have Sickness(disease,ds).

DATA POSSESSION POLICY

Is trusted (t) allowed to have/possess the data group Sickness(disease,ds) ? (Y if Yes)

SAVE DATA POSSESSION POLICY & CLOSE

Choose an entity: trusted (t) | Choose a data group: Sickness(disease,ds) | Choose a data type:

**Verification result (privacy violation):**

message2trust

trusted

VERIFICATION RESULT

THE ARCHITECTURE DOES NOT PRIVACY CONFORM WITH THE POLICY (SEE DETAILS BELOW)

PRIVACY VIOLATION OF THE POLICY: trusted CAN HAVE THE DATA (OF TYPE) : sickness(disease,ds)

