

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

User manual

Nov 2021

By Vinh T. Ta

© All rights reserved

Contents

Introduction	5
How to run the tool.....	5
Supported Operating Systems	5
Software Dependencies	5
The main features	5
Policy specification highlight.....	6
Architecture specification highlight.....	6
GUI-based architecture specification.....	6
Text-based architecture specification.....	7
Finally, in Figure 4 we summarise the features of architecture specifications.	8
Menu points and tabs	8
GUI features	10
Saving a GUI-based architecture.....	11
TEXTMODE features.....	12
Verification options.....	12
Architecture specification under the GUI mode.....	13
Actions	19
Components/entities	21
Data types	21
Data protection policy specification	26
Saving a policy.....	26
Entities/Components (the top policy pane).....	27
Data groups/Data types (the middle policy pane).....	28
Sub-policy specification (the bottom policy pane)	29
The data collection sub-policy	30
The data usage sub-policy.....	31
The data storage sub-policy	31
The data transfer sub-policy	32
The data retention sub-policy.....	32
The data possession/data access sub-policy	33
The data connection (permit) sub-policy.....	34
The data connection (forbid) sub-policy.....	34
Conformance verification:	36
Functional conformance	36
Violation of the functional conformance.....	36

Privacy conformance.....	37
Violation of the privacy conformance.....	37
DPR conformance	37
Violation of the DPR conformance	38
Application Examples:.....	39
GUI-MODE: Example without using cryptographic functions.	39
Example 1.....	39
Example 2.....	45
GUI-MODE: Examples with cryptographic functions	48
Example 1.....	48
Example 2.....	51
Example 3.....	54
Example 4.....	55
Example 5.....	57
Example 6.....	61
Example 7.....	64
Example 8/a	66
Example 8/b	68
Example 8/c.....	69
Example 9/a	71
Example 9/b	74
Example 10.....	75
Example 11.....	78
Example 12/a	80
Example 12/b	83
Example 12/c.....	84
Example 13 (Pseudonym application).....	85
TEXT-MODE EXAMPLES.....	89
Example 1 (LINK)	89
Example 2 (LINK)	90
Example 3 (LINK)	91
Example 4 (LINK)	91
Example 5 (LINKUNIQUE)	91
Example 6 (LINKUNIQUE)	93
Example 7 (LINKUNIQUE)	93
Example 8 (Collection Consent).....	94

Example 9 (Transfer Consent)	95
Example 10 (action CALCULATEFROM/ CALCULATEFROMAT).....	97
The Case of Attackers.....	99
External Attackers.....	101
Insider Attackers	102
Hybrid Attackers	104

Introduction

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

The official GitHub page of the tool where the updates and version information can be found is the following: <https://github.com/Dataprove/Dataprovetool/>

The official website of the tool is currently hosted at: <https://sites.google.com/view/dataprove/>

How to run the tool

The tool is available in .py format. You need to install **Python version 3.8.2 or 3.8.5 32bit/64bit or above** (from <https://www.python.org/downloads/>). If you have an older version of Python (e.g., Python 3.6), the file still runs, but some functionality might be missing.

How to run the .py file under Windows 7/10:

1. Double click on the file, or
2. `python "<replace_with_path_to_file>\DataProVe-prototype.py"`

How to run the .py file under Linux or macOS (Note: the current version has not been tested thoroughly in Linux and macOS yet):

1. `python "<replace_with_path_to_file>/DataProVe-prototype.py`, or
`python3 "<replace_with_path_to_file>/DataProVe-prototype.py`.

Supported Operating Systems

DataProVe was implemented and only tested on Windows 7/10. The current version of the tool has not been tested on macOS or Linux yet.

Software Dependencies

The tool requires:

- Python 3.8 or higher
- The tkinter package (python-tk) that supports GUI features.
- Further packages required: time, webbrowser, json, itertools, re.

The main features

In this section, we highlight the main features of the tool. Throughout the paper, we will use the terms “component” and “entity”, interchangeably.

Policy specification highlight

The policy specification page enables the specification of eight sub-policies (data collection, usage, storage, transfer, deletion, data possession and data connection (forbid and permit)), as can be seen in Figure 1.

PROVIDE A NEW ENTITY: PROVIDE A DESCRIPTION:

PROVIDE A GROUP OF DATA TYPES: IS THIS UNIQUE? THE DATA TYPES IN THIS GROUP:

Choose an entity Choose a data group Choose a data type

SUB-POLICIES :

Data Collection	Data Usage	Data Storage	Data Retention	Data Transfer	Data Possession	Data Connection (Permit)	Data Connection (Forbid)
-----------------	------------	--------------	----------------	---------------	-----------------	--------------------------	--------------------------

Figure 1. The policy specification page.

Architecture specification highlight

The tool supports both **GUI-based** and **Text mode** options for architecture specification, while the policy specification is only GUI-based. The user can save the policy and architecture at any time, and later open them to modify, extend or just run the verification.

GUI-based architecture specification

In the GUI mode (the ARCHITECTURE-GUI tab), the user can specify the architecture using graphical features. An example GUI based specification of an architecture can be seen in Figure 2.

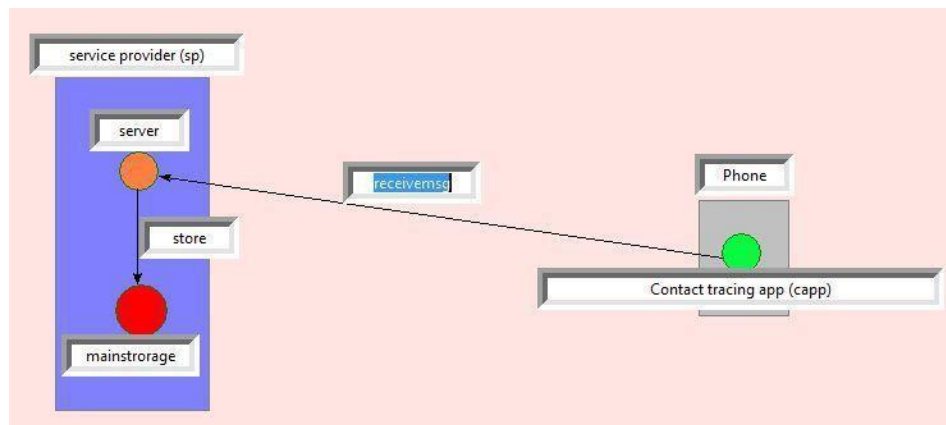


Figure 2. An example GUI-based specification of an architecture.

Text-based architecture specification

In the “text mode” (the ARCHITECTURE-TEXTMODE tab), the user can specify the architecture using a text editor. This option can be useful in case of large architectures, or the user does not want to spend time on drawing graphical elements. Figure 3 shows an example textual specification of the architecture.

TEXT EDITOR FOR ARCHITECTURE SPECIFICATION (PLEASE PROVIDE ONE ARCH. ACTION PER ROW)

```

CALCULATEFROMAT (panel, ephIDa, seed, Time (t) )
CALCULATEFROMAT (server, ephIDb, seed, Time (t) )

```

SAVE CONTENT

(AFTER OPENNING AN ARCHITECTURE: Before running the conformance verification the user must click on SAVE CONTENT.)

Figure 3. An example textual specification of an architecture.

Finally, in Figure 4 we summarise the features of architecture specifications.

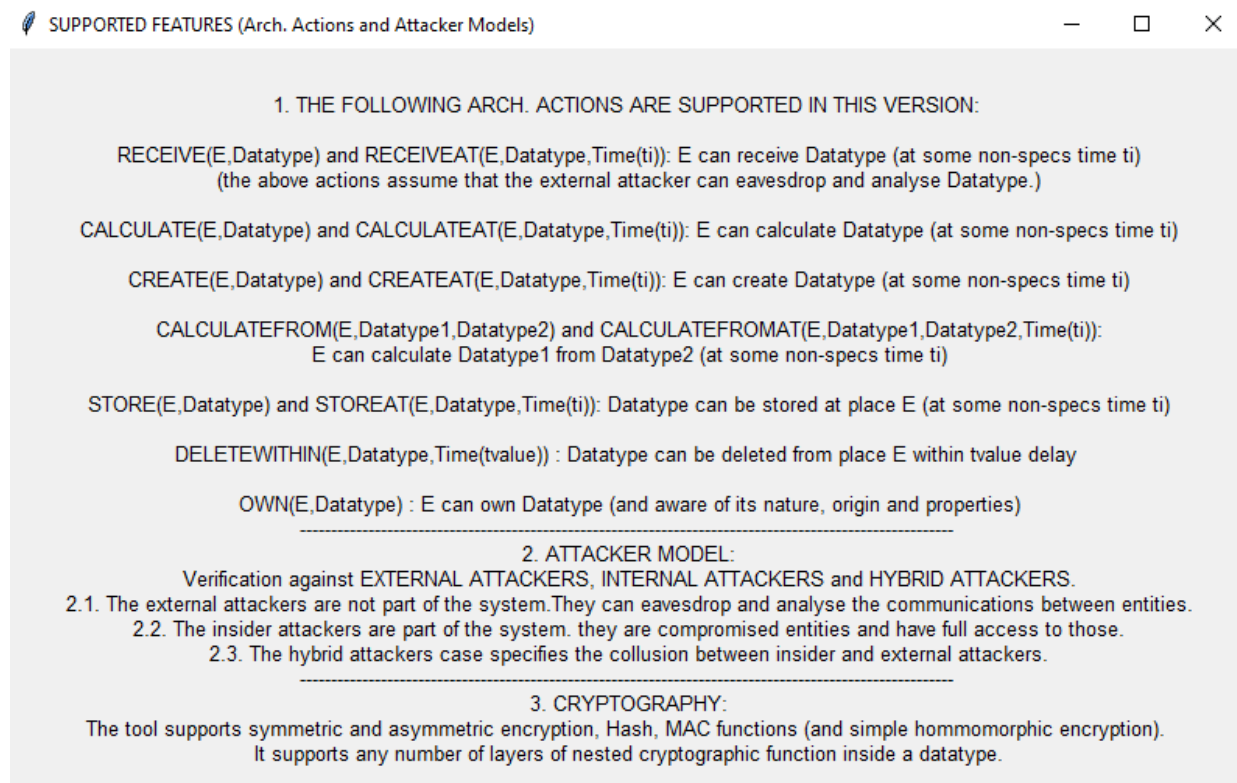


Figure 4. Feature summary for the architecture specification.

Menu points and tabs

In this section, we present and highlight the tabs that can be found at the top-left corner of the architecture specification pane.

After launching the tool, as depicted in Figure 5, the default page can be seen, where the user can specify the architecture.

The menu bar includes the main options "POLICY", "ARCHITECTURE-GUI", "ARCHITECTURE-TEXTMODE" and "VERIFY", for policy specification, architecture specification (GUI and TEXT modes) and conformance verification between them, respectively.



Figure 5. After launching the app, the system architecture specification page can be seen.

Specifically, under the option “POLICY”, the user can specify a new data protection policy, save the policy, and open a saved policy.

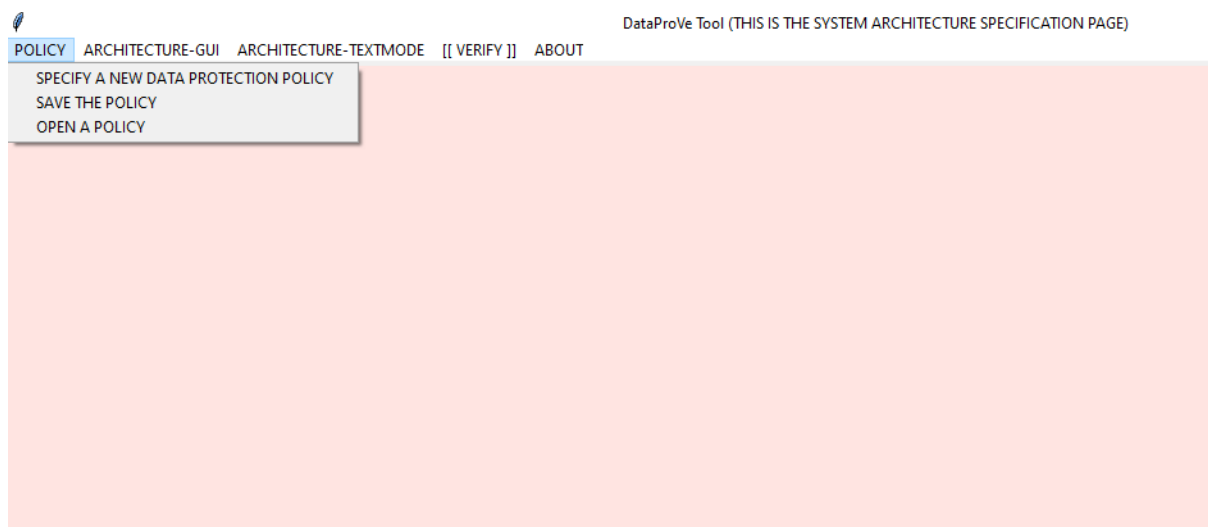


Figure 6. The Policy tab.

Under the tab ARCHUTECHTURE-GUI, the user can specify an architecture using GUI-based elements and the GUI features, outlined in Figure 7.

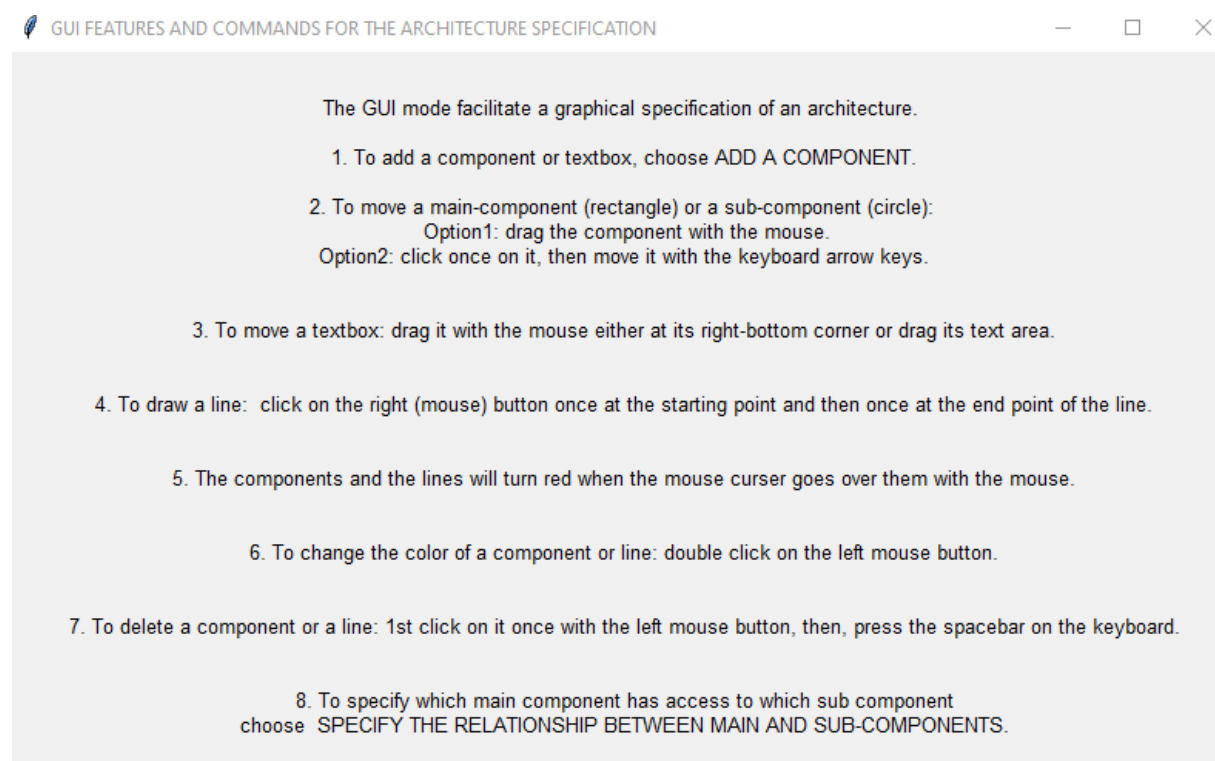


Figure 7. GUI features for the GUI-based architecture specification.

GUI features

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

The main difference between the option SAVE THE ARCHITECTURE and the option PRINT ARCHITECTURE TO FILE is that while in the first case, all the GUI elements and the settings are saved, the last option only save the actions defined in an architecture.

The option “SPECIFY THE RELATIONSHIP BETWEEN THE MAIN AND SUB-COMPONENT (GUI)” allows the user to specify which entity/component can have access to which other entities/components. If an entity/component E1 can have some data D, and an entity/component E2 can have access to E1, then E2 can also have D. The same is valid in case of data linkability or data connection.



Figure 8. The GUI based architecture specification options. The user can specify an architecture by adding visual elements like main components as boxes, sub-components as circles, and lines as connections.

Saving a GUI-based architecture

Under the ARCHITECTURE-GUI mode, an architecture can be saved with the extension *.arch, as can be seen in Figure 9.

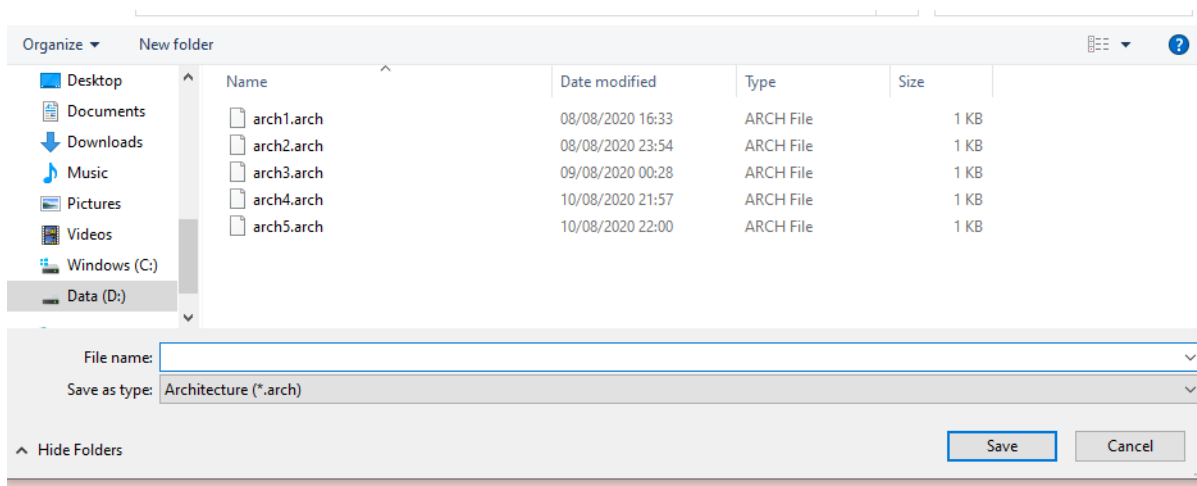


Figure 9. Saving a GUI-based architecture.

TEXTMODE features

The user can specify the architecture in textual mode as well. In this case, the text editor can be opened, and the architectural actions can be typed in row by row. This feature can be useful in case of large architectures with a huge number of actions, as well as when the user wants to save time on drawing components and lines.

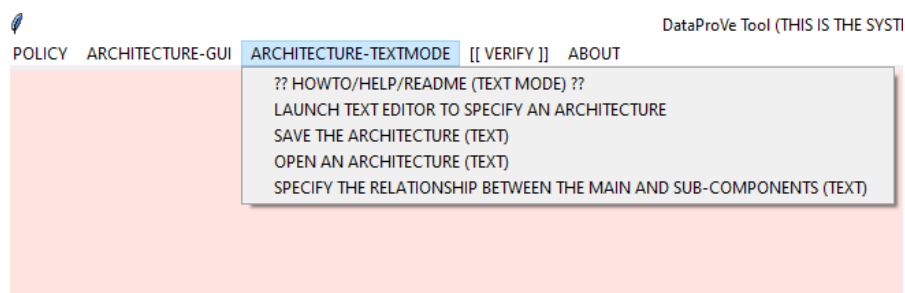


Figure 10. The Policy tab.

The tabs and options under the textual mode are as follows (see Figure 10): The first tab “?? HOWTO/...??” shows the guideline and hints on the usage of the TEXT MODE. The second tab “LAUNCH TEXT EDITOR ...” is for launching a text editor for providing the architecture in textual format. The next two tabs are for saving and opening an architecture in textual format. The architecture file saved in this case will get a .txt extension. Finally, the last tab is for specifying which entity can have access to which other entities (which is similar to the GUI case, but under textual mode).

Verification options

Finally, under the tab “VERIFY”, we can select an option to verify the conformance between our specified policy and architecture (as shown in Figure 10).

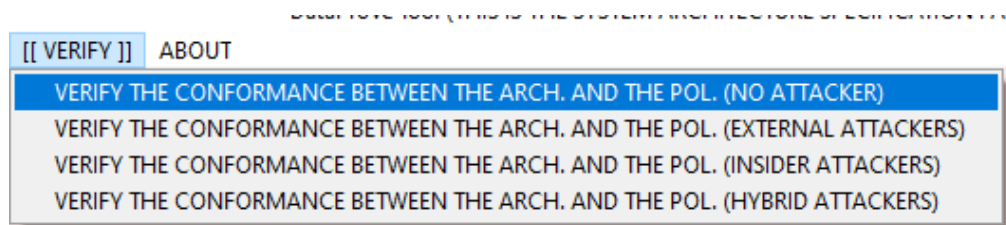


Figure 11. Verification options.

There are four options here: “NO ATTACKER”, “EXTERNAL ATTACKERS”, “INSIDER ATTACKERS”, and “HYBRID ATTACKERS”.

- The first case runs the verification in a genuine environment (without the presence of an attacker).
- The second case runs the verification in the presence of the external attacker(s) who can eavesdrop on the communication, and can analyse the data (hence, access and link data types). The verification returns whether the external attacker(s) can have/access certain data type or link two data types.
- The third case runs the verification in the presence of the insider attackers. The insider attackers are compromised entities who are under control of the attackers. An attacker “inherits” the ability of a compromised entity and can have/access and link data that the entity would be able to do. The situation is more interesting when several compromised entities can collude and share data with each other. The tool also considers this case.
- Finally, the fourth option runs the verification against the hybrid attacker(s), which is basically the case of collusion between the external attackers and the insider attackers.

Architecture specification under the GUI mode

In this section, we explore the architecture specification under the GUI MODE.

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

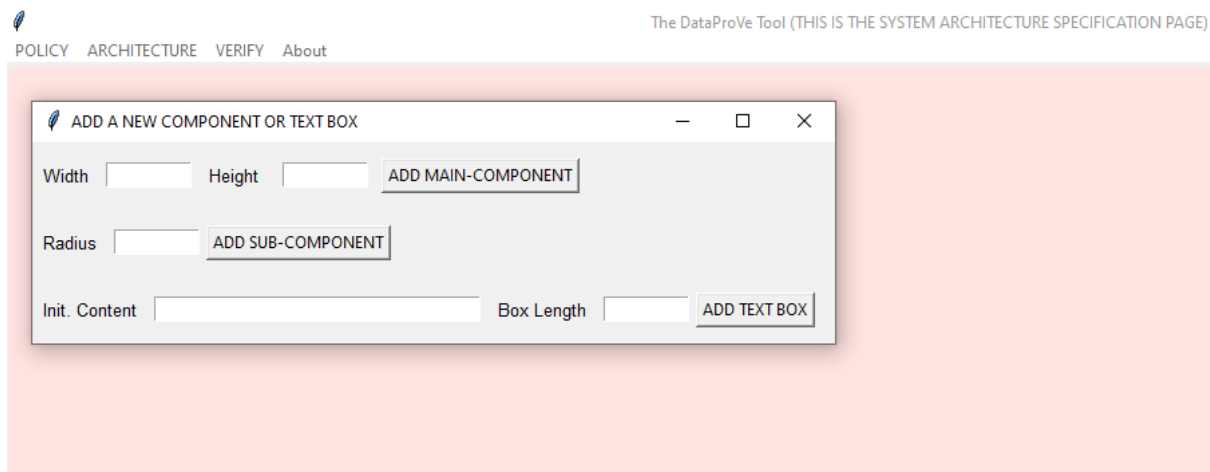


Figure 12. Add a new component or a text box.

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, as two main components can share a sub-component and only one main-component has access to the data stored at the sub-component. 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 13-15.

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.

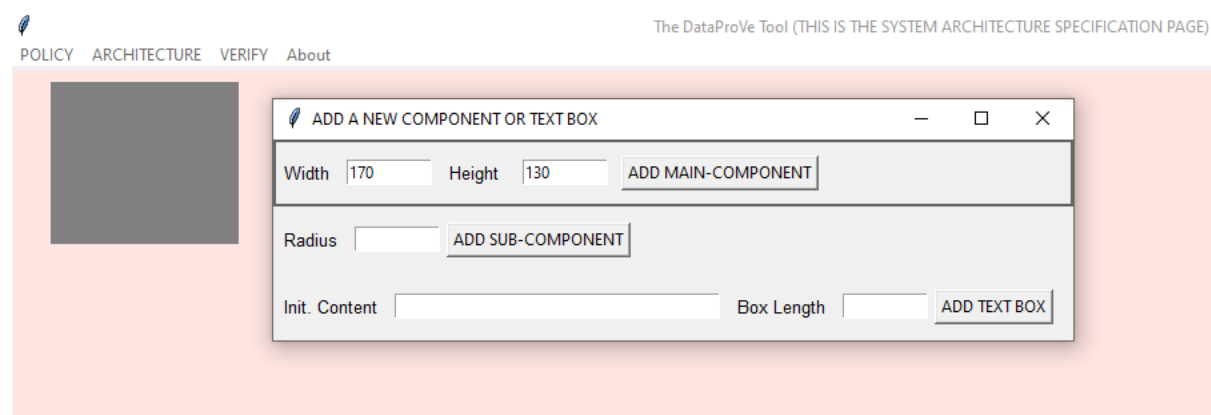


Figure 13. Adding a main component of size 170x130.

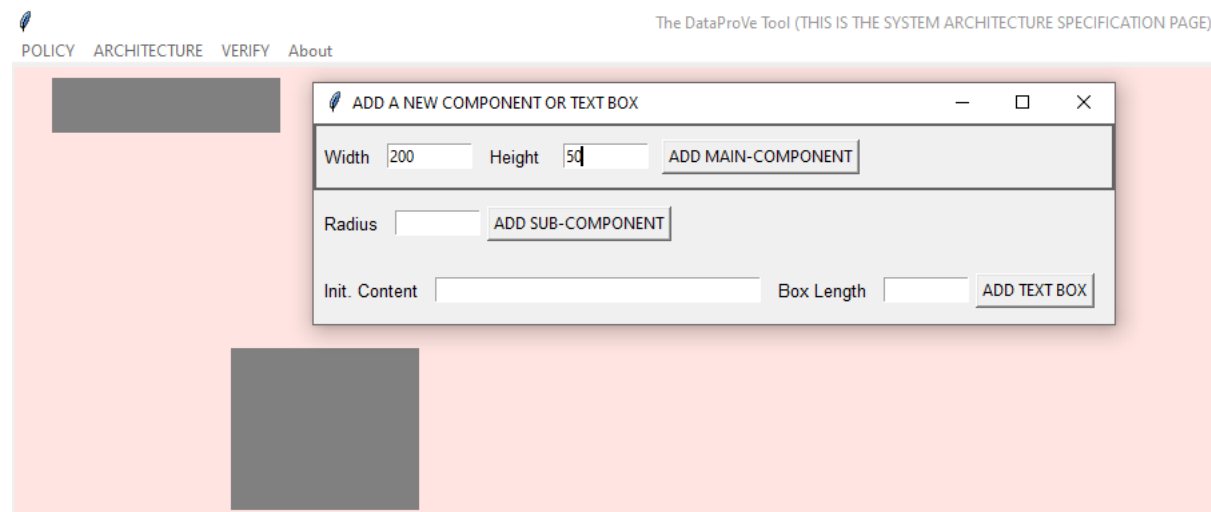


Figure 14. Adding a new main component of size 200x50.

The users can use a 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.

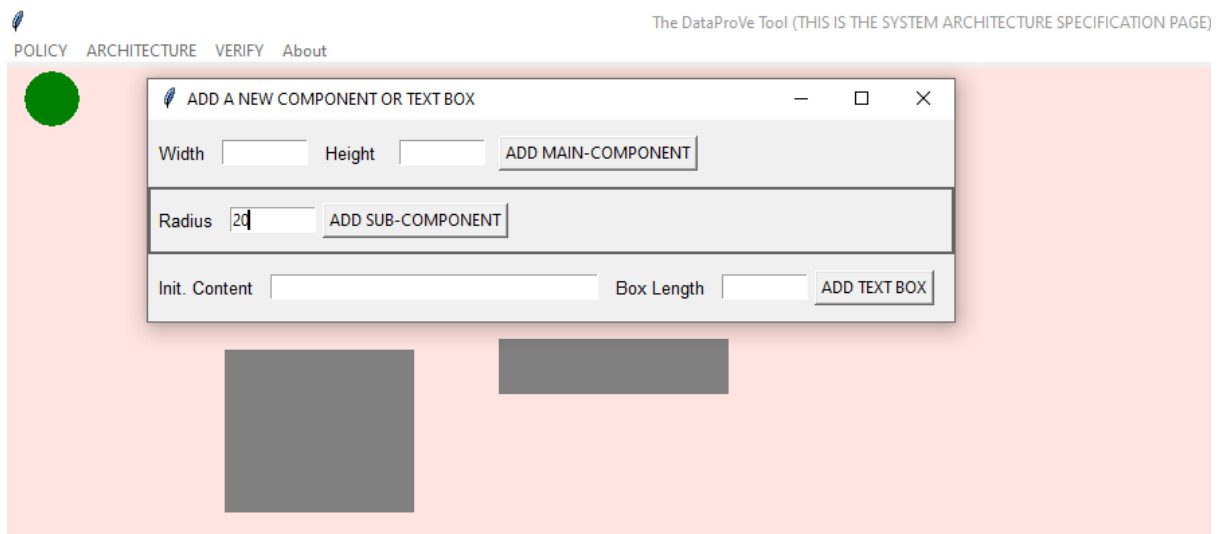


Figure 15. Adding a new sub-component with a radius size of 20.

To change the colour of an object, double click on with the left mouse button, then choose a preferable colour as shown in Figure 16.

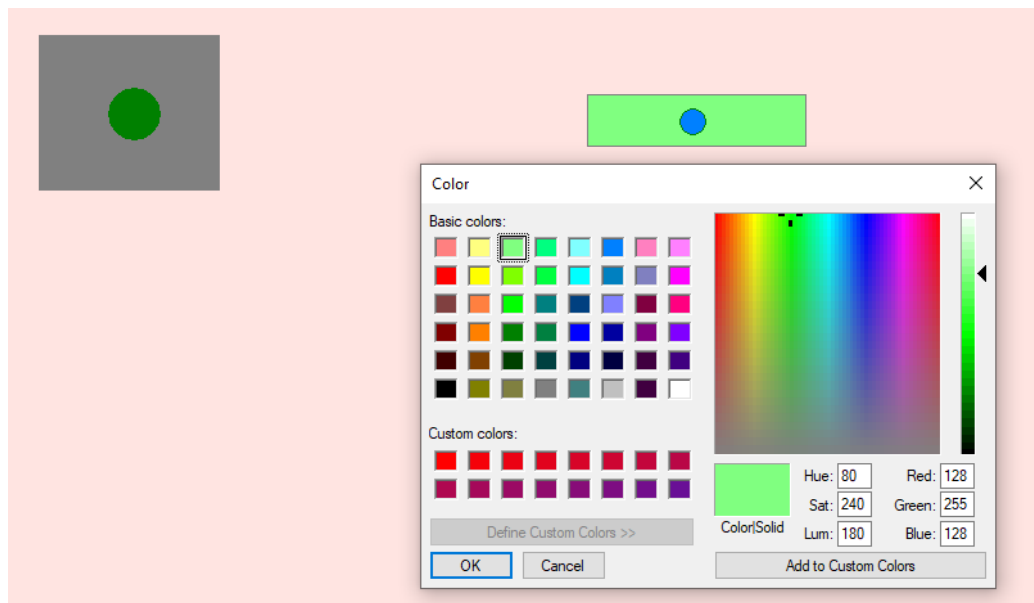


Figure 16. Change the colour of the objects (main- and sub-components).

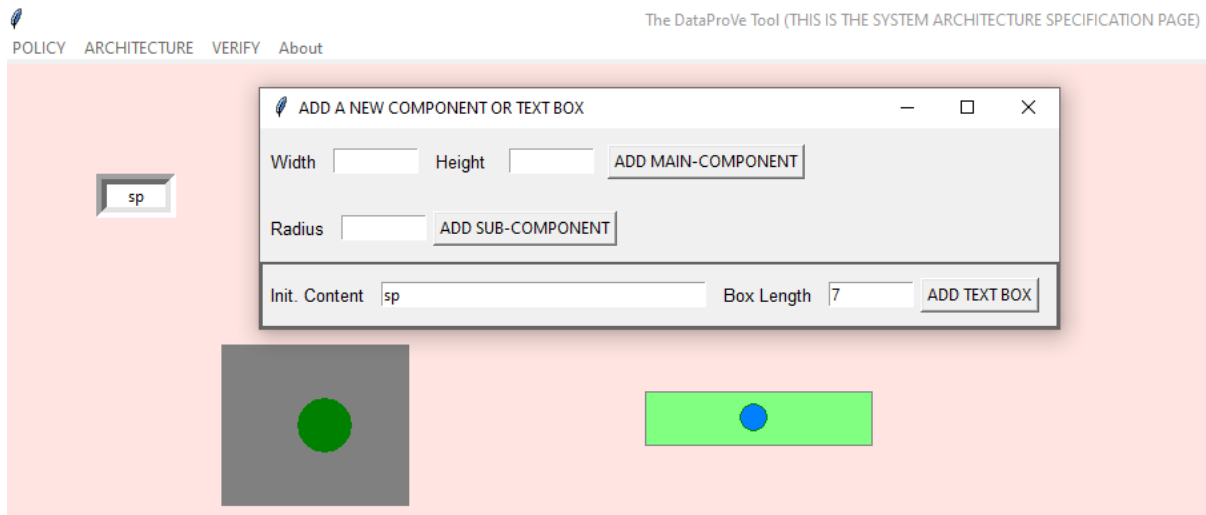


Figure 17. Add description (name) of the objects (sp, 7).

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 17.).

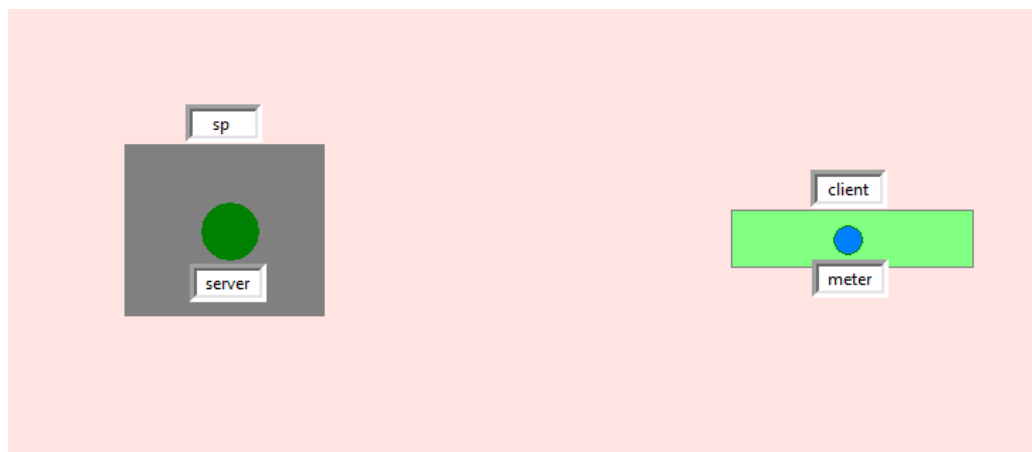


Figure 18. Assigned names for each main and sub-component.

As depicted in Figure 18, 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-components or other main-components. An example can be seen in Figures 19-20, between sp, server and meter.

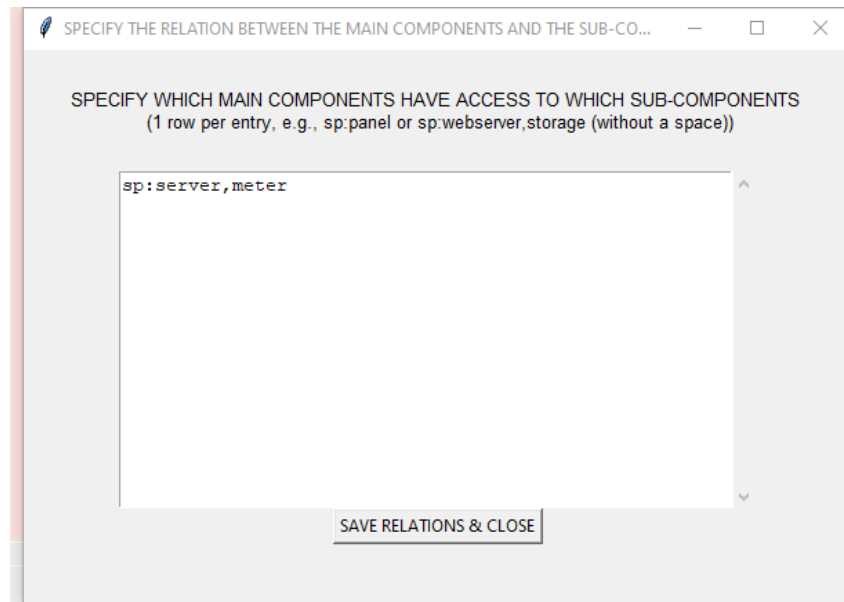


Figure 19. Specifying which main component has access to the data in which sub-component or other main-components (sp has access to server and meter).

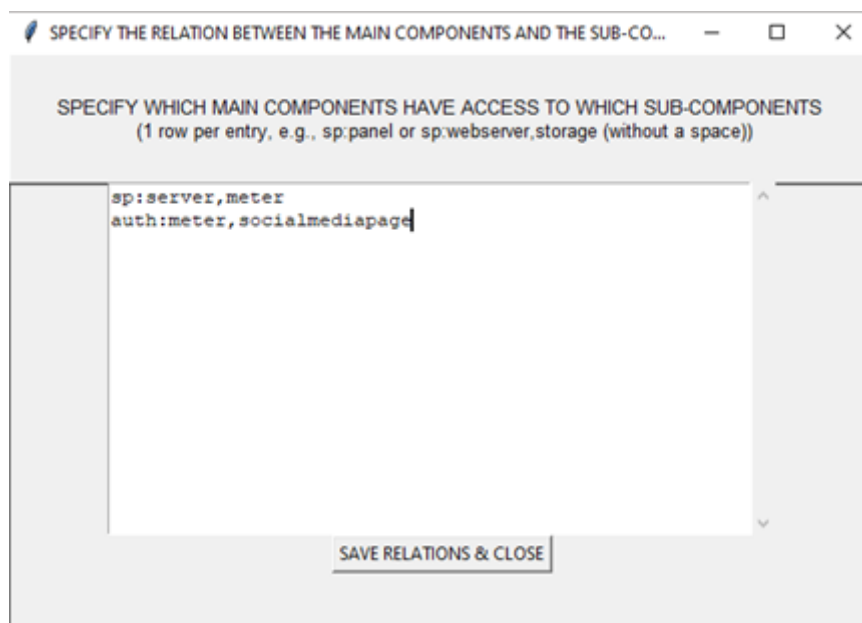


Figure 20. sp (service provider) has access to server and meter, auth (third party authority) has access to meter and a social media webpage. This specification is also used to model compromised entities or components, for the verification against insider and hybrid attackers (options 3-4 in Figure 11).

Note that in the relationship specifications (as in Figures 19-20), any component can be on the right-side of a relationship above, not only the sub-components of a component (except for the same component on the left side of the colon, to avoid recursive effect). 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 most mouse button and drag from one component to another (a component can be either main or sub).

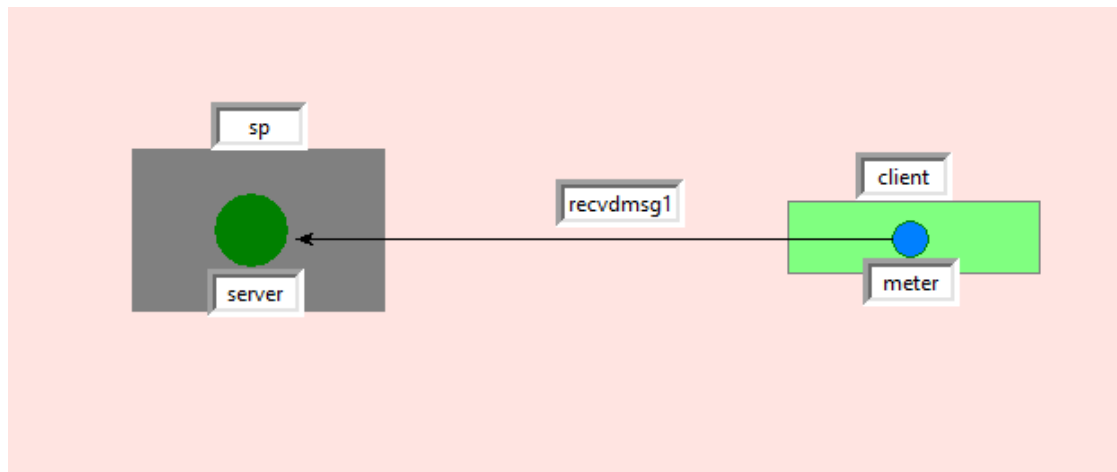


Figure 20. Draw an arrow from the component meter to server.

In Figure 16, 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 17, the content says that sp can receive a reading that contains the energy consumption (energy) and the customer ID (custID).

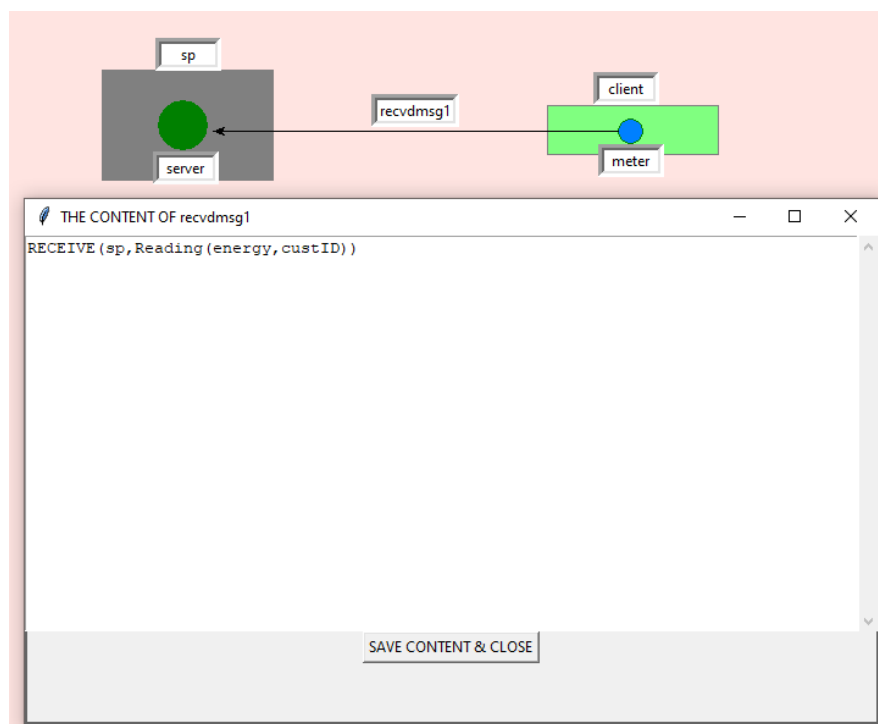


Figure 21. Specify the message content of recvdmsg1.

To delete an object (a component, text box, or an arrow), the user should click once on the object so that it turns red colour, and thereafter, hit the space bar on the keyboard.

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 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", "CALCULATEFROM", "CALCULATEFROMAT", "STORE", "STOREAT", "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)).

In the following, the reserved/pre-defined keywords are highlighted in bold, while the “non-bold” arguments 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). If a component owns a piece of data of type Datatype, then we assume that it is also aware of its nature, origin, or properties.
- **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. This action captures a data transmission via a “public channel”, which means that the external attacker can capture the content and analyse it offline (see the second verification option in Figure 11).
- **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 is received. A consent data type can be
 - **Cconsent**(Datatype,component), for collection consent
 - **Uconsent**(Datatype,component), for usage consent
 - **Sconsent**(Datatype,component), for storage consent
 - **Fwconsent**(Datatype,component), for transfer consent.Similar to the **RECEIVE** case, this action captures a data transmission via a “public channel”.

- **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)**).
- **CALCULATEFROM**(component,Datatype1, Datatype2):
This action defines that a component can calculate a piece of data of type Datatype1 from Datatype2, for instance, **CALCULATEFROM**(sp,id,randomseed) defines that a service provider sp can calculate an ID from a piece of data of type random seed.
- **CALCULATEFROMAT**(component, Datatype1, Datatype2,**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, **CALCULATEFROMAT**(sp,id,randomseed,**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/containing a name, address and phone number can be stored in the main storage of the service provider at some non-specific time **t**.
- **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 numerical time values such as 3y (3 years) or 2y (2 years), 6m (6 months).

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

Components/entities

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**, **attacker**, **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).
- **att**: this reserved keyword defines the attacker (that can represent an external, insider, or hybrid attacker, or a group of colluding attackers).
- **trusted**: this reserved keyword defines a trusted authority that can 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`, and `nhsnumber`.
- **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:

1. This version of DataProVe supports **any number of layers** of nested **cryptographic** operation, and data types. In addition, it supports **any number of layers** of non-crypto operation/data type.
2. The compound data types contain data types that belong to the **same** living individual.

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

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

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

- **Cconsent**(Datatype,component): This is the type of data collection consent on a piece of data of type Datatype, and component is given consent to carry out certain actions on the data of type Datatype. For example, **Cconsent**(sicknessrec,phone), and **Cconsent**(Account(creditcard,address),backend) capture the collection consent on the illness information, and the account containing a credit card number and address.
- **Uconsent**(Datatype,component): The type of usage consent on a piece of data of type Datatype, and component is given consent to carry out certain action on the data of type Datatype. For example, **Uconsent**(Energy(gas,water,electricity),sp), **Uconsent**(address,clientpc).
- **Sconsent**(Datatype,component): The type of storage consent on a piece of data of type Datatype and component is given consent to store the data. E.g., **Sconsent**(personalinfo,mainstorage), **Sconsent**(Account(creditcard,address),phone) define the types of storage consent on a type of personal information and account, which are given consent to be stored in the main storage place of the service provider, and phone, respectively.
- **Fwconsent**(Datatype,component): The type of transfer consent on a piece of data of type Datatype, and a component to whom the data can be transferred (i.e. a component who can receive this data). E.g., **Fwconsent**(personalinfo,auth), **Fwconsent**(Account(creditcard,address),backend) defines the type of transfer consent on the type of personal information and account, respectively, as well as a

third party authority (auth) or backend server to which the given data is transferred, respectively.

- The type of time: $\text{Time}(t)$ or $\text{Time}(tvalue)$, where $\text{Time}(t)$ is the type of non-specific time, and $\text{Time}(tvalue)$ is the type of numerical time. The pre-defined special keywords $t, t1, t2, \dots, tn$, for some natural n , denote the type of non-specific time, and $tvalue$ is a type of time value (such as 5 years, 2 hours, 1 minute, etc.).

$tvalue$ 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**, **CALCULATEFROMAT**, **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,sp),**Time**(t))
The service provider can receive a collection consent on illness information at some non-specific time t . Instead of t , we can also use $t1, t2, \dots, tn$, to denote those different actions can happen at different non-specific times.
- **RECEIVEAT**(sp,Uconsent(Webpage(photo,job),server),**Time**(t))
The service provider can receive a usage consent on a webpage at some non-specific time t . Here, server has the right to do something with the webpage. Similarly, $t1, t2, \dots, tn$ can be used instead of 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 ($t1, t2, \dots, tn$ can be used instead of t).
- **CREATEAT**(server,Account(name,address),**Time**(t)):
The service provider can create an account that contains a name and address at some non-specific time t . Similarly, $t1, t2, \dots, tn$ can be used instead of t .
- **CALCULATEAT**(sp,Bill(tariff,Energy(gas,water,electricity)),**Time**(t)):
The service provider can create an account that contains a name and address at some non-specific time t ($t1, t2, \dots, tn$ can be used instead of 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 data type.

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 type 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 be a data type or a component. Pseudonym is a means for achieving a certain degree of privacy in practice, as the real identity 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 the 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 the previous case, but also includes the time construct Time(t).
- The types of cryptographic primitives and functions: DataProVe supports the basic cryptographic primitives for architectures. Again, in the following, we write the reserved keywords in bold letters.
 - Private key: **Sk**(Pkeytype)
This data type defines the type of private key (secret key) used in asymmetric encryption algorithms. Its argument has a type of public key (Pkeytype). We note that Pkeytype is not a reserved data type. Basically, a secret key of type **Sk**(Pkeytype) corresponds to the public key of type Pkeytype.
 - 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 (of type Datatype) and a symmetric key (of type 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)) 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)) specifies that a service provider can calculate a private key corresponding to the public key (of type pkey).
 - **OWN**(sp,pkey) 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)) 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).

Data protection policy specification

In this section, we detail the features and options for policy specification. We explore the options under the tab “specify a new data protection policy”, and highlight how to save a policy, so that the user can return and continue working on the specification.

Saving a policy

First of all, under the Policy tab (see Figure 23) at the top-left corner of the architecture specification pane, a policy can be saved in a file with the extension *.pol, which can be opened later.

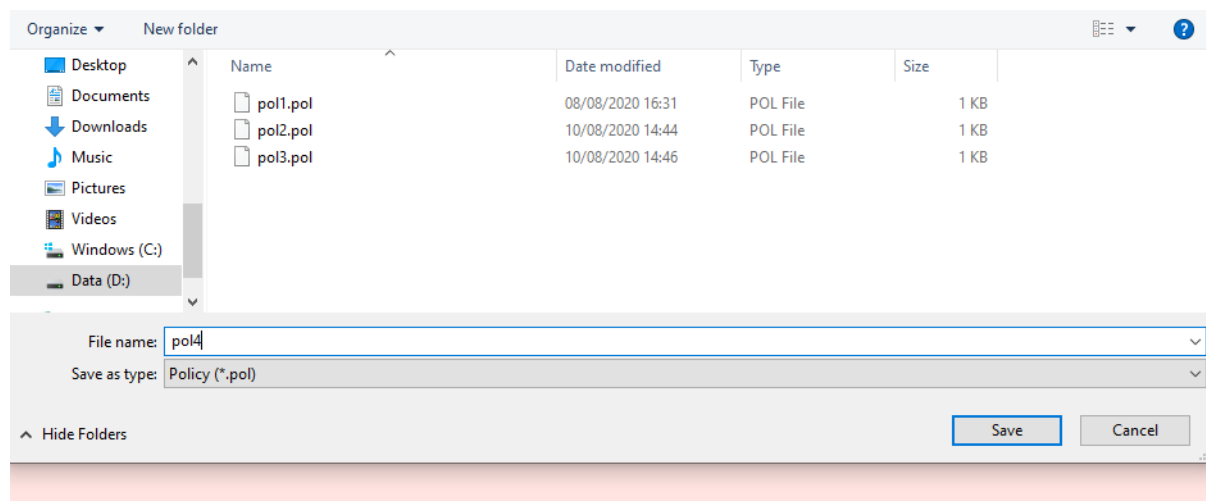


Figure 22. Saving a policy.

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

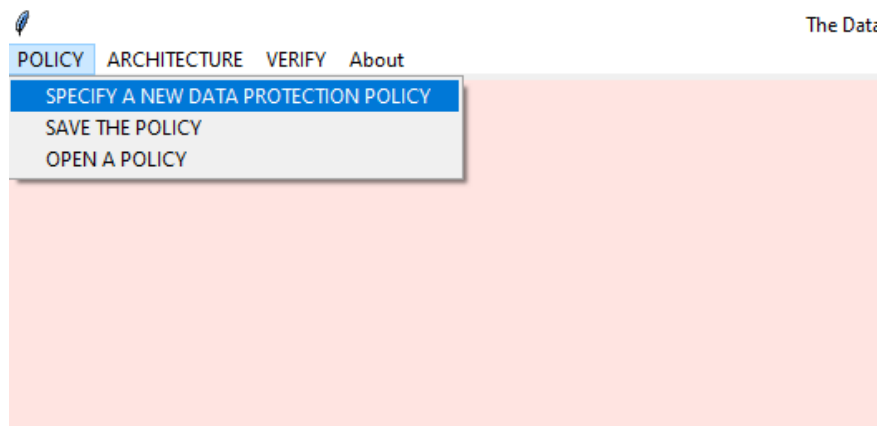


Figure 23. Options under the policy tab.

The policy specification page is divided into three parts.

- The top part is for specifying entities/components that are parts of a service.
- The middle part is for specifying the data types and groups of data types that follow the same policy.
- Finally, the bottom part is for specifying the eight sub-policies.

Entities/Components (the top policy pane)

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 of an entity, and on the right side, the full name/description to help identifying the meaning of the notation. For instance, in Figure 24, 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***

The DataProVe Tool (THIS IS THE DATA PROTECTION POLICY SPECIFICATION PAGE)

PROVIDE A NEW ENTITY: PROVIDE A DESCRIPTION:

PROVIDE A GROUP OF DATA TYPES: IS THIS UNIQUE? THE DATA TYPES IN THIS GROUP:

Choose an entity: Choose a data group: Choose a data type: SPECIFY EACH SUB-POLICY:

Figure 24. The Policy specification page.

(service provider), and att (attacker) are pre-defined entities that are already added by default (hence, the user does not need to add them). The user can specify any other entities.

Data groups/Data types (the middle policy pane)

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 25, a data group denoted by *personalinfo* is defined which includes four data type, *name*, *address*, *dateofbirth*, and *phonenumber*.

The screenshot shows a light blue interface with three main sections. On the left, under the label "PROVIDE A GROUP OF DATA TYPES:", there is a text input field containing the text "personalinfo". In the center, under the label "IS THIS UNIQUE?", there are two buttons: "Yes" (highlighted in blue) and "No". On the right, under the label "THE DATA TYPES IN THIS GROUP:", there is a scrollable list box containing the text "name", "address", "dateofbirth", and "phonenumber". To the far right, there is a button labeled "ADD DATA GROUP & TYPES".

Figure 25. 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 group together with its types cannot be used to uniquely identify an individual, so the option “No” was chosen.

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 light blue interface with three main sections. On the left, under the label "PROVIDE A GROUP OF DATA TYPES:", there is a text input field containing the text "energy". In the center, under the label "IS THIS UNIQUE?", there are two buttons: "No" (highlighted in blue) and "Yes". On the right, under the label "THE DATA TYPES IN THIS GROUP:", there is a scrollable list box containing the text "gas", "water", and "electricity". To the far right, there is a button labeled "ADD DATA GROUP & TYPES".

Figure 26. Specifying data groups (*energy*) and its data types.

The screenshot shows a light blue interface with three main sections. On the left, under the label "PROVIDE A GROUP OF DATA TYPES:", there is a text input field containing the text "name". In the center, under the label "IS THIS UNIQUE?", there are two buttons: "No" (highlighted in blue) and "Yes". On the right, under the label "THE DATA TYPES IN THIS GROUP:", there is an empty scrollable list box. To the far right, there is a button labeled "ADD DATA GROUP & TYPES".

Figure 27. Specifying only data type (*name*) that is not unique.

For instance, in Figure 27, we only define a data type called name, which cannot be used to uniquely identify an individual. This case, we left the text box on the right empty. In Figure 28, for example, a data type of insurance number, *insurancenumbe*, is specified that can be used to uniquely identify an individual.

Figure 28. Specifying only data type (insurance number) that is unique.

Sub-policy specification (the bottom policy pane)

A data protection policy is defined on a data group/type and an entity/component (we recall that the terms component and entity are used interchangeably).

As mentioned before there are two pre-defined entities by default, sp (service provider) and att (attacker) can be seen in Figure 29.

Figure 29. List of entities and the default entities att and sp.

In DataProVe, each policy consists of eight sub-policies (Figure 30), 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. 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 31).

SUB-POLICIES :	Data Collection	Data Usage	Data Storage	Data Retention	Data Transfer	Data Possession	Data Connection (Permit)	Data Connection (Forbid)

Figure 30. The policy specification page (sub-policies).

Figure 31. The policy specification page (select data groups and entities).

The data collection sub-policy

The data collection sub-policy consists of two parts. The top part is for specifying if consent is required to be collected for a given data type. E.g., for insurance numbers in Figure 32.

Figure 32. The data collection sub-policy specification.

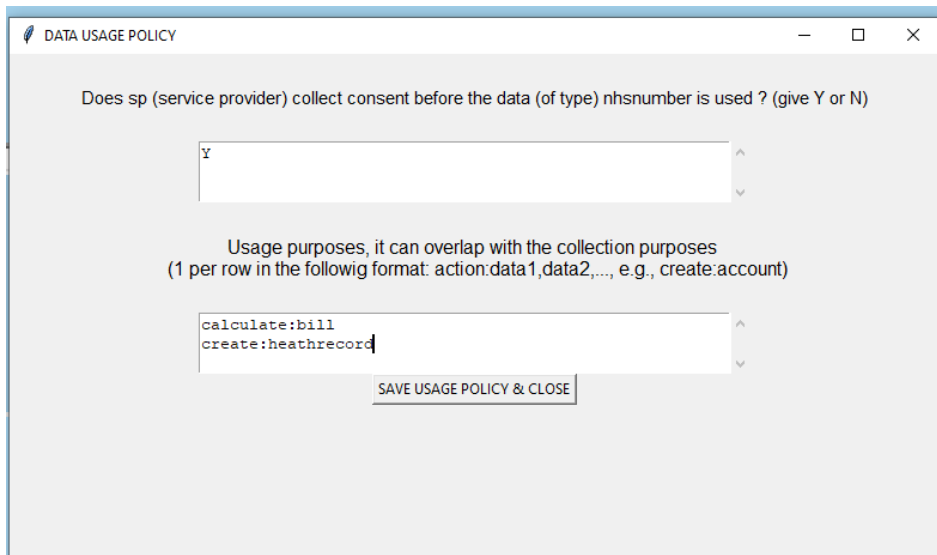
In the bottom part, the collection purposes can be given row by row, each row with a different action in the format of (again, no space is allowed between two words). Namely:

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 32, 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 data 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 it is related to the consent and purposes for data usage. For instance, in Figure 33, 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.

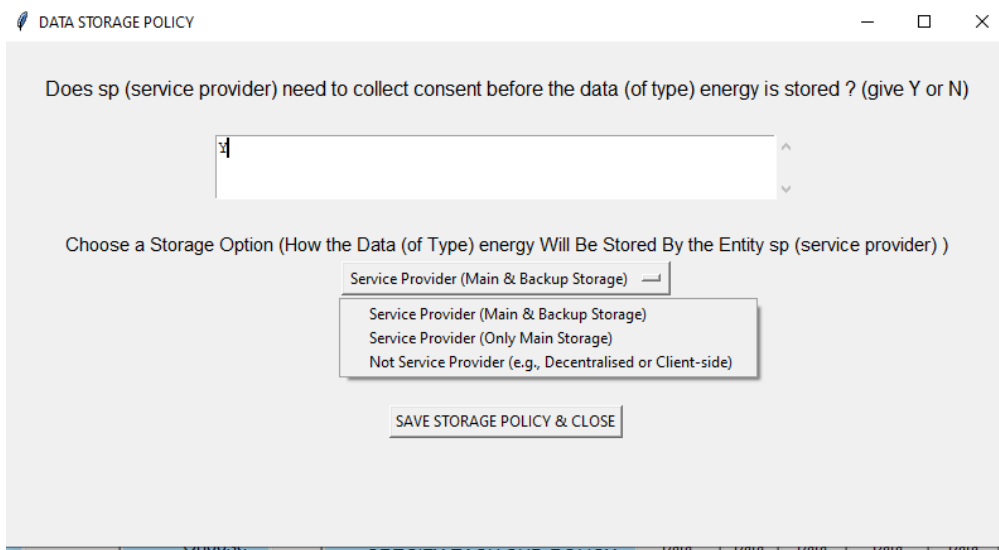


The screenshot shows a web form titled "DATA USAGE POLICY". It contains a question: "Does sp (service provider) collect consent before the data (of type) nhsnumber is used ? (give Y or N)". Below the question is a text input field containing the letter "Y". Another question follows: "Usage purposes,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

Figure 33. 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 (Figure 34). Like 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).



The screenshot shows a web form titled "DATA STORAGE POLICY". It contains a question: "Does sp (service provider) need to collect consent before the data (of type) energy is stored ? (give Y or N)". Below the question is a text input field containing the letter "Y". Another question follows: "Choose a Storage Option (How the Data (of Type) energy Will Be Stored By the Entity sp (service provider))". Below this is a dropdown menu with four options: "Service Provider (Main & Backup Storage)", "Service Provider (Main & Backup Storage)", "Service Provider (Only Main Storage)", and "Not Service Provider (e.g., Decentralised or Client-side)". At the bottom of the form is a button labeled "SAVE STORAGE POLICY & CLOSE".

Figure 34. The data storage sub-policy specification.

The storage sub-policy offers three pre-defined options, “Service Provider (Main & Backup Storage)”, “Service Provider (Only Main Storage)”, “Not Service Provider”. The first option means that the data must 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 must only be stored in the main storage places, without any back up. Finally, the third option is when the data must be stored at the client devices or decentralised storage places that are not under control of the service provider.

The data transfer sub-policy

This sub-policy defines how a piece of data of a data type/group is transferred. 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 can be transferred.

Figure 35. 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 sub-policy, the users have to define the storage sub-policy first, as the deletion will be defined in the storage places specified in the storage sub-policy. A reminder is shown to the user if the storage sub-policy has not been specified before the retention sub-policy.

Figure 36. A data storage sub-policy is closely related to the deletion sub-policy.

Figure 37. We set the retention delays in each storage place (10y+6mo in the first case and 15y in the second).

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.

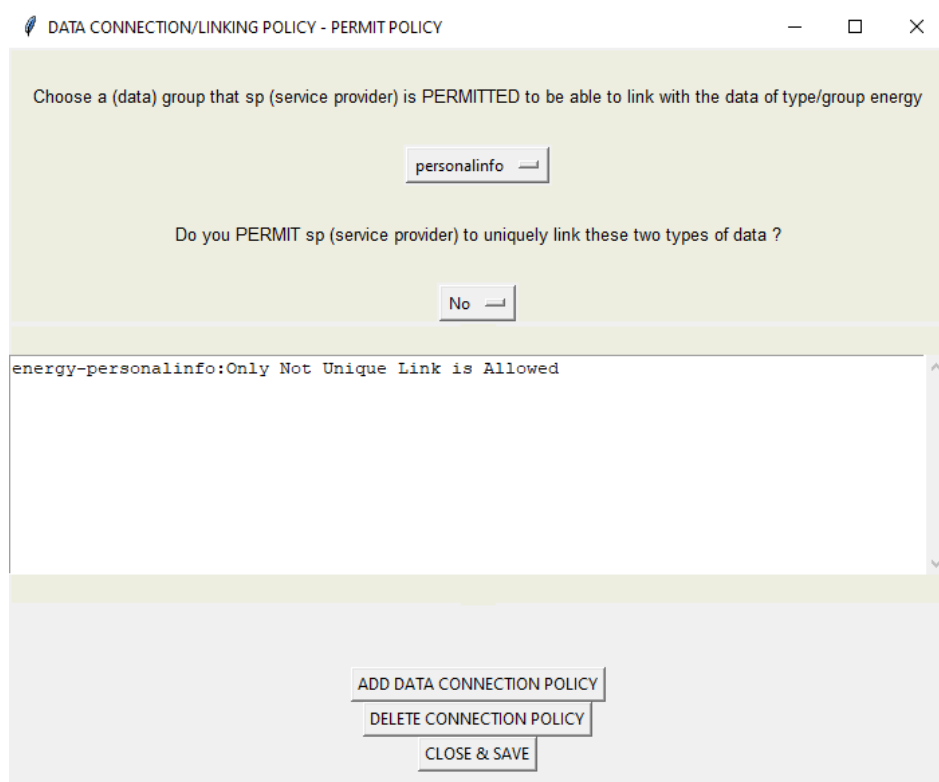
The data possession/data access sub-policy

The data possession sub-policy defines who can have/access a piece of data of a given group. The users only need to specify which entities are allowed to have or access a given data group, DataProVe will automatically assume that the rest entities/components are not allowed to have/access the selected type of data (see Figure 38 for example).

Figure 38. The data possession sub-policy for energy. We give auth the right to access the data type energy.

The data connection (permit) sub-policy

This sub-policy specifies which entity is permitted (has the right) 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 39. The data connection permission sub-policy. Here we give sp the right to link energy with personal info.

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 39, we specified that the service provider (sp) is permitted to link the data group energy and the data group *personalinfo*. However, we do not give sp the right to uniquely link these two data groups. Obviously, if *personalinfo* was defined as unique, then unique link is always possible, so there is a chance that the architecture always violates this requirement of the policy.

The data connection (forbid) sub-policy

This sub-policy is the counterpart of the permit policy. While in case of the data possession sub-policy, the user only needs to specify which entity is allowed to have or access certain type of data, and DataProVe automatically assumes that the rest specified entities are not allowed, in case of linkability, the user needs to explicitly forbid the right for an entity to link a pair of data types/groups.

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 40. The data connection (permit) sub-policy. The case when only unique link is forbidden.

For example, in Figure 40, we forbid the right for the service provider (sp) to link the data group *personalinfo* with the data group *energy*. Here, we also forbid the unique link of the two data groups for sp.

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 41. The data connection (permit) sub-policy.

Conformance verification:

To verify the conformance of a specified architecture and specified policy, as mentioned before, there are four options “NO ATTACKER”, “EXTERNAL ATTACKERS”, “INSIDER ATTACKERS”, and “HYBRID ATTACKERS”.

In the “NO ATTACKER” case, the verification will cover all the eight sub-policies, while in case of the rest three (attackers’ cases), only the data possession and data connection sub-policies are verified. Specifically, we only verify if the attacker(s) can have/access and link certain types of data, data pairs.

We define three types of conformances, 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 give the right for an entity to have/access 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
5. STOREAT action defined for only mainstorage, and for the same data type/group.
6. 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,datatype,Time(t)) or RECEIVE(ent,datatype).

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(datatype,comp),Time(t)), or RECEIVEAT(sp,Sconsent(datatype,comp),Time(t)), or

- RECEIVEAT(sp,Uconsent(data,comp),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 the right for an entity to have or access a piece of data of certain type/group, then in the architecture the same entity cannot have or access a piece of data of the same type/group.
2. If in the policy, we forbid the right for an entity 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 the right for an entity to have or access a piece of data of certain type/group, but in the architecture the same entity can/is be able to have or access a piece of data of the same type/group.
2. In the policy, we forbid the right for an entity to link/uniquely link two pieces of data of certain types/groups, but in the architecture the same entity can 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.
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:datatype* 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:datatype* 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 DELETEWITHIN(**mainstorage**,data,Time(value)) 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 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 “toy” examples. We can specify our architecture either with or without cryptographic functions. In the following, first we present an example that does not include any cryptographic functions, then we provide some examples with different cryptographic functions (including nested layers of crypto functions), and afterwards, we highlight the examples for each sub-policy.

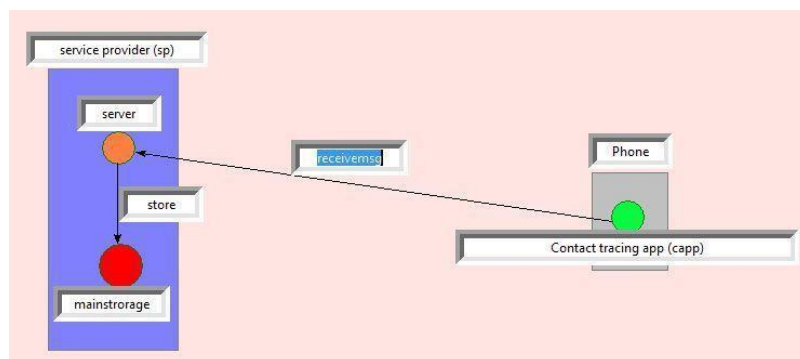
The policy and architecture files used in these examples can be found in the **“Pol and arch files used in the manual v.0.9.8.zip” file.**

GUI-MODE: Example without using cryptographic functions.

Example 1

In this example, a service provider (sp) collects positive (virus) test records sent by contact tracing apps. A record contains a unique ID (*id*) and a set of places (*places*) where the phone has been brought to, and the record is stored in the main storage place(s) of sp.

The architecture level: The architecture is defined as follow:



Where the content of *receivemsg* is as follow (we discuss both the application of the action with or without the Time(*t*) construct, starting with the non-timed actions first):

Content of receivemsg:

- **RECEIVE(server,Positivetest(id,places))**

The content of *store* is as follow:

Content of store:

- **RECEIVE**(mainstorage,Positivetest(id,places))
- **STORE**(mainstorage,Positivetest(id,places))

Alternatively, we would get the same verification result for the corresponding actions with the **Time(t)** construct.

Content of receivemsg:

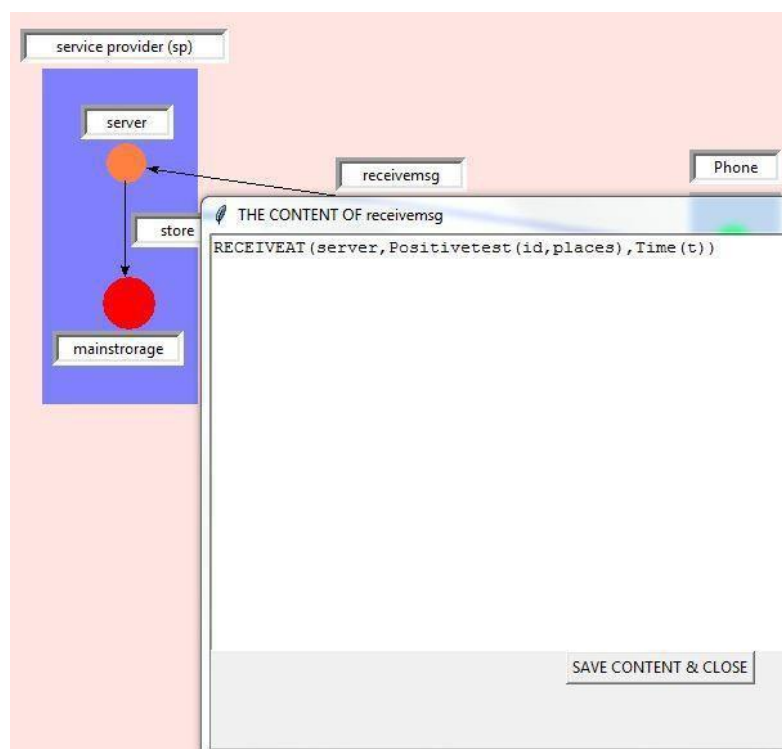
- **RECEIVEAT**(server,Positivetest(id,places),**Time(t)**)

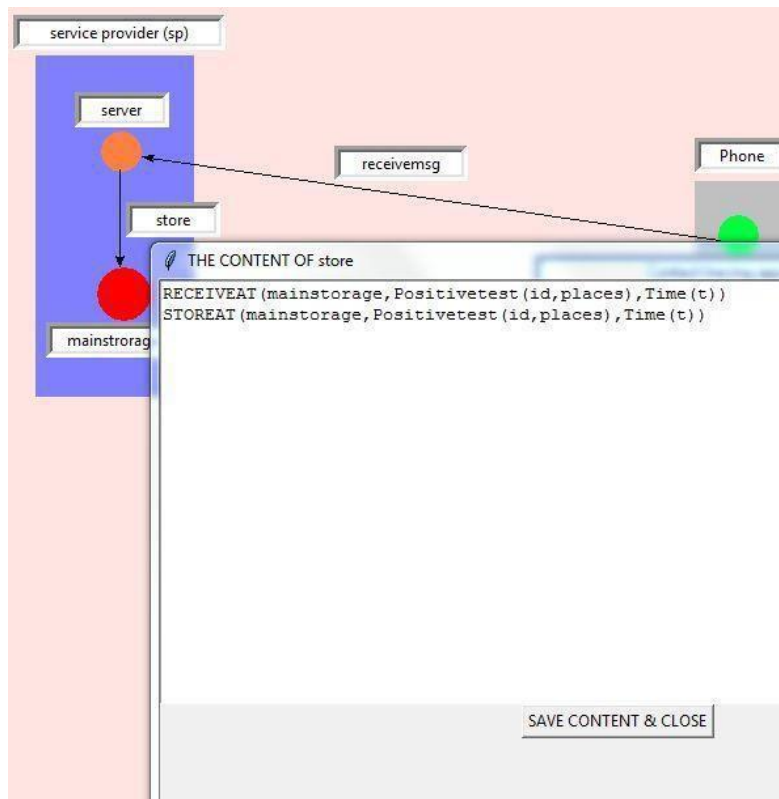
The content of (the text box) *store in the figure above is as follow:*

Content of store:

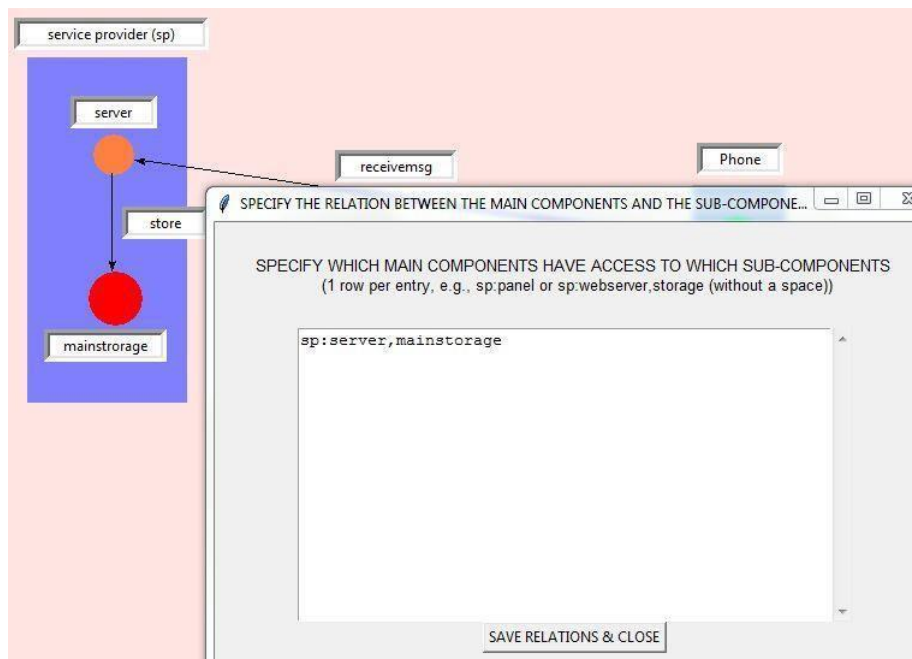
- **RECEIVEAT**(mainstorage,Positivetest(id,places),**Time(t)**)
- **STOREAT**(mainstorage,Positivetest(id,places),**Time(t)**)

As shown on the architecture specification page in DataProVe:





HasAccessTo: In the architecture, we also set that the service provider has access to the data stored/handled/collected by its sub-components, server and mainstorage.



The policy level:

Policy option 1: In the policy, we define/add three data groups, namely, *id*, *places*, and *name*.

(For simplicity, in the following examples, we only use the field “group of data types” to define data types. The field “data types in this group” is left blank/empty, however, as presented in Figures 19-20, the user can specify data groups that contain several data types.)

PROVIDE A NEW ENTITY:

PROVIDE A DESCRIPTION:

ADD NEW ENTITY

PROVIDE A GROUP OF DATA TYPES:

IS THIS UNIQUE?

THE DATA TYPES IN THIS GROUP:

ADD DATA GROUP & TYPES

Choose an entity

Choose a data group

Choose a data type

SUB-POLICIES :

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

Where:

Choose an entity

Choose a data group

id

name

places

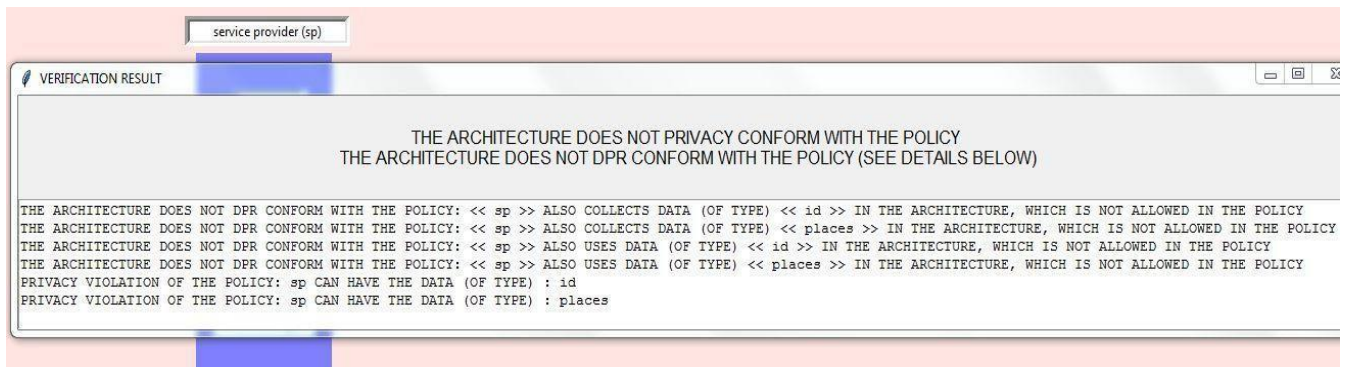
Choose a data type

SUB-POLICIES :

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

For the first attempt, we do not specify/set anything in the sub-policies. So, by default this means that *id*, *places*, and *names* are not allowed to be collected and used by *sp*, as well as *sp* does not have the right to possess/have any of *id*, *name*, *places*.

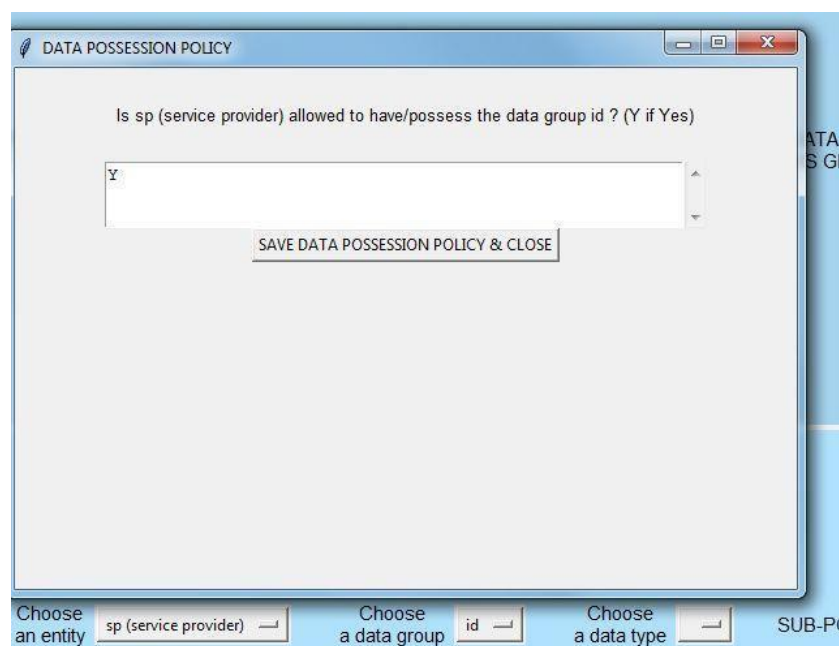
As the result, we got the following:



In the last two lines of the verification result, we can see that sp can have both id and places, which violates the data possession sub-policy. However, there is no privacy violation detected for name.

The first four lines of the results capture the violation of the DPR conformance, because we neither specify the collection and usage purposes for id nor places. Note that in this version of DataProVe, the actions STORE/STOREAT also correspond to usage (besides CALCULATE/CALCULATEAT and CREATE/CREATEAT).

Policy option 2: Next, in the policy, let us set sp to have the right to possess/have the data of group (type) id and places.



DATA POSSESSION POLICY

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

Y

SAVE DATA POSSESSION POLICY & CLOSE

Choose an entity: sp (service provider)

Choose a data group: places

Choose a data type:

SUB-POLICY

After running the verification, as a result, we got the following: The last two lines of the verification result shows a functional conformance as in accordance with the policy, sp can have the data of types id and places in the architecture.

service provider (sp)

VERIFICATION RESULT

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) << id >> 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) << places >> 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) << id >> 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) << places >> IN THE ARCHITECTURE, WHICH IS NOT ALLOWED IN THE POLICY

The architecture functionally conform with the policy: sp can have the data (of type) id

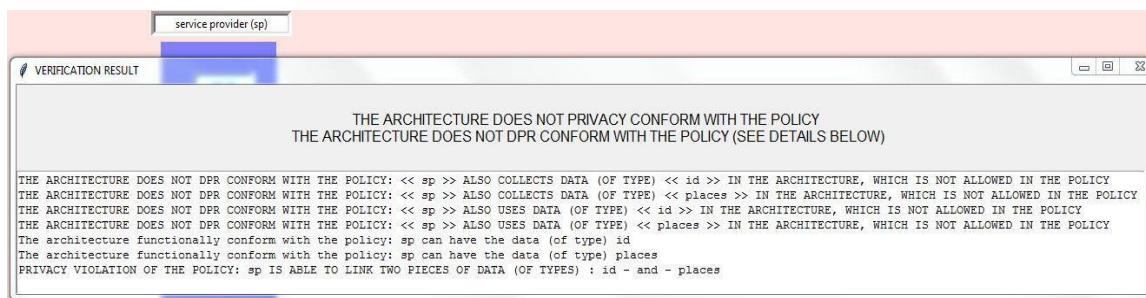
The architecture functionally conform with the policy: sp can have the data (of type) places

Policy option 3:

Finally, for the last case, in the policy, we set that sp does not have the right to link *id* with *places* (id-places):



After running the verification, as the result, we got the following: The last line of the verification result shows the violation of the privacy conformance relation, as *sp* can link the data of types *id* and *places* in the architecture.

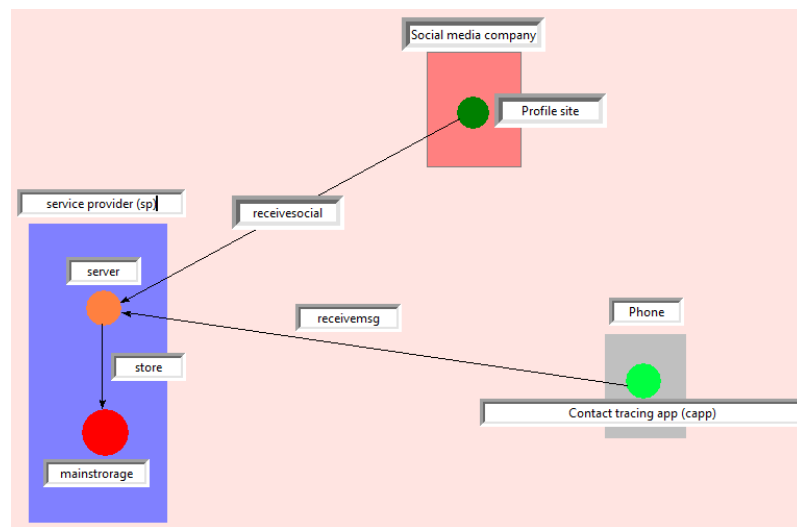


Example 2

In the second version of the contact tracing app example, a social media company (as main-component) is defined in the architecture from which the service provider (*sp*) can get the social profile that contains *name* and *places*, where *name* is the name of the person in the social profile (*Socprofile*), and *places* are the places that have been posted on this social media site (showing that the named person has visited those places).

The architecture level:

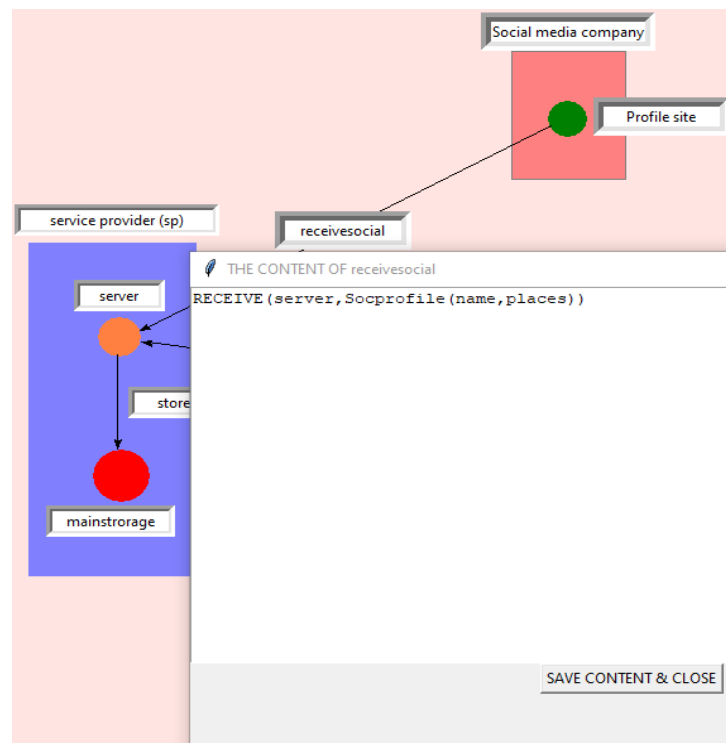
The architecture is defined as follow:



Where the content of *receivesocial* is as follow:

Content of receivesocial:

- **RECEIVE(server,Socprofile(name,places))**



The policy level:

In the policy, sp is forbidden/does not have the right to link *name* and *places* and forbid the sp to (be able to) link *id* and *name*. Otherwise, the rest sub-policies are left blank/empty.

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

name

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

No

id-name:Any Link is Forbidden

ADD DATA CONNECTION POLICY
DELETE CONNECTION POLICY
CLOSE & SAVE

Choose an entity sp (service provider) Choose a data group id Choose a data type SUB-POLICIES :

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

places

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

No

name-places:Any Link is Forbidden

ADD DATA CONNECTION POLICY
DELETE CONNECTION POLICY
CLOSE & SAVE

Choose an entity sp (service provider) Choose a data group name Choose a data type SUB-POLICIES :

As a result, we get:

```
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) << id >> 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) << places >> 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) << id >> 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) << places >> IN THE ARCHITECTURE, WHICH IS NOT ALLOWED IN THE POLICY
PRIVACY VIOLATION OF THE POLICY: sp CAN HAVE THE DATA (OF TYPE) : places
PRIVACY VIOLATION OF THE POLICY: sp CAN HAVE THE DATA (OF TYPE) : name
PRIVACY VIOLATION OF THE POLICY: sp CAN HAVE THE DATA (OF TYPE) : id
PRIVACY VIOLATION OF THE POLICY: sp IS ABLE TO LINK TWO PIECES OF DATA (OF TYPES) : name - and - places
PRIVACY VIOLATION OF THE POLICY: sp IS ABLE TO LINK TWO PIECES OF DATA (OF TYPES) : id - and - name
```

The last two lines of the verification result show that we got the violation of the privacy conformance relation, as in the architecture, *sp* can link name and places, as well as *id* and *name*.

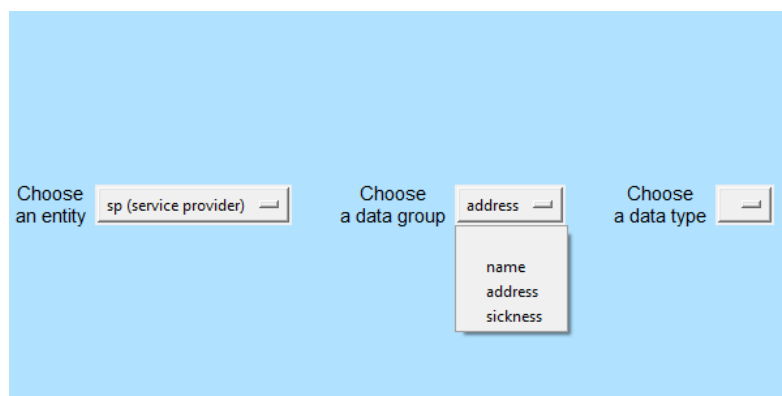
GUI-MODE: Examples with cryptographic functions

Example 1

See “Examples-GUI/Example-Crypto1” in the “Pol and arch files used in the manual v.0.9.8.zip”.

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

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



Then, we leave the data passion 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 sp (service provider) Choose a data group address Choose a data type

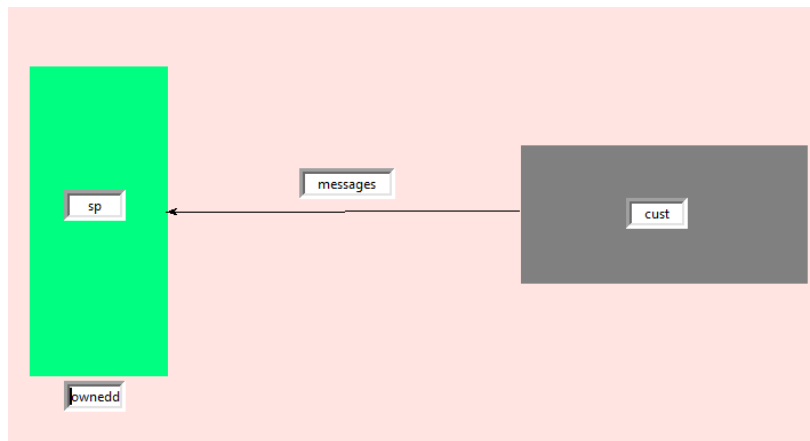
DATA POSSESSION POLICY

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

SAVE DATA POSSESSION POLICY & CLOSE

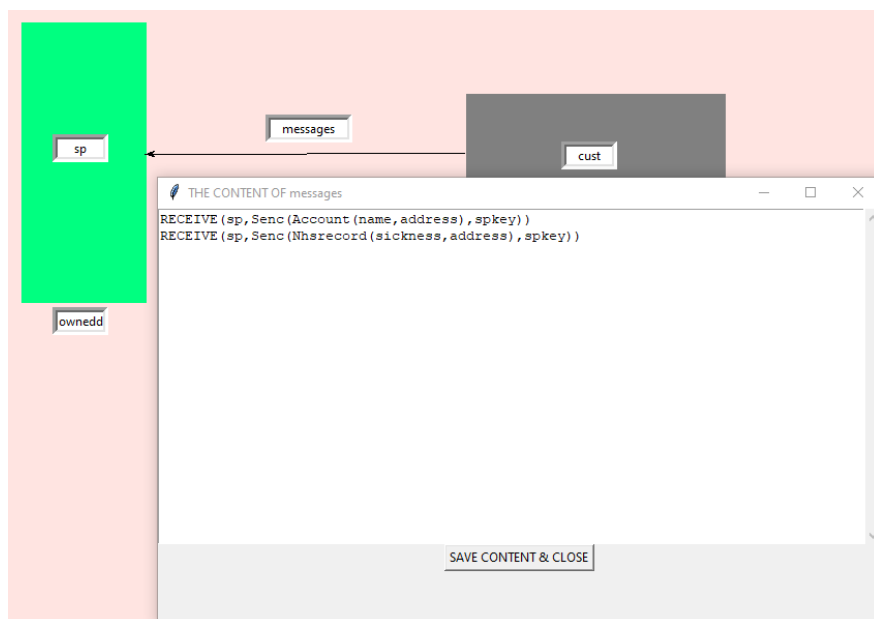
Choose an entity sp (service provider) Choose a data group name Choose a data type

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* contain 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.



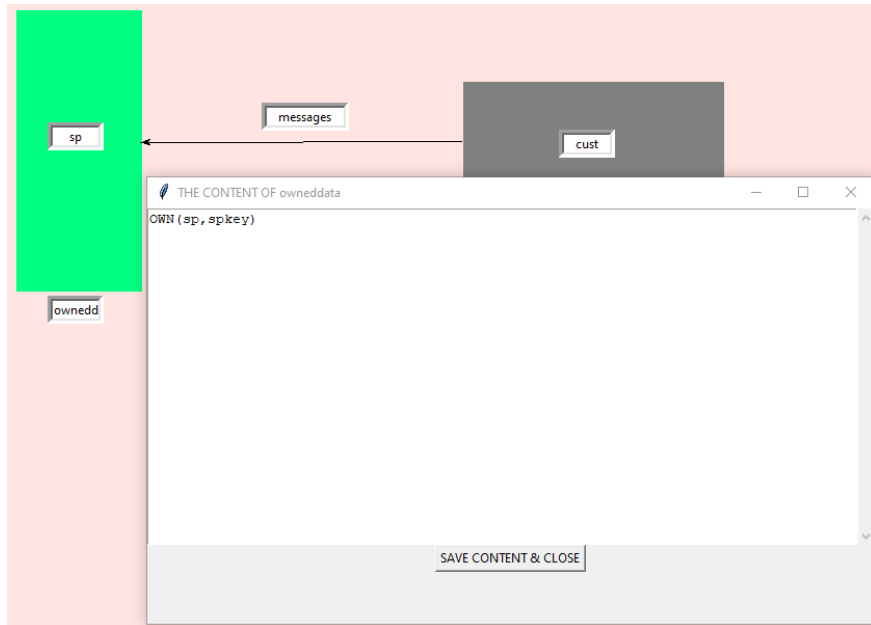
In *messages*, we specify two RECEIVE actions, saying that sp can receive an encrypted account that contains a name and an address inside it. The encryption is symmetric encryption using the shared key spkey.

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

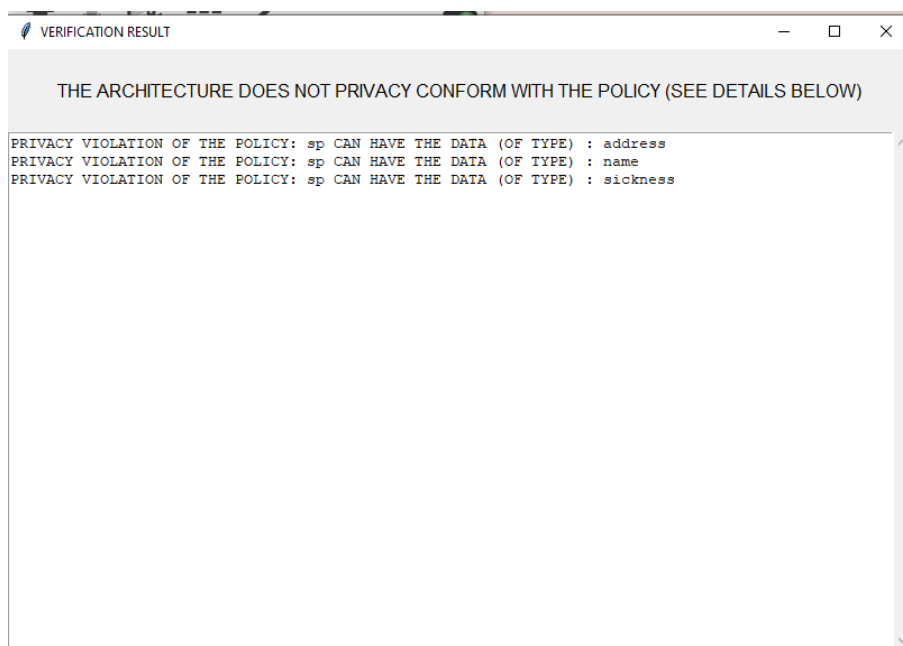


In *owneddata*, we specify an action OWN, saying that sp can own a data of type share key, spkey.

OWN(sp,spkey)



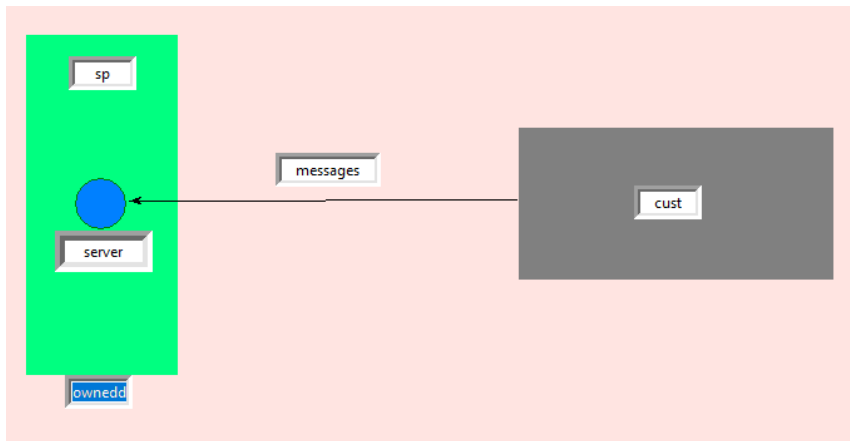
Then, we run the verification, and as result we got privacy violation as the service provider can have both three data groups that we forbid in the policy.



Example 2

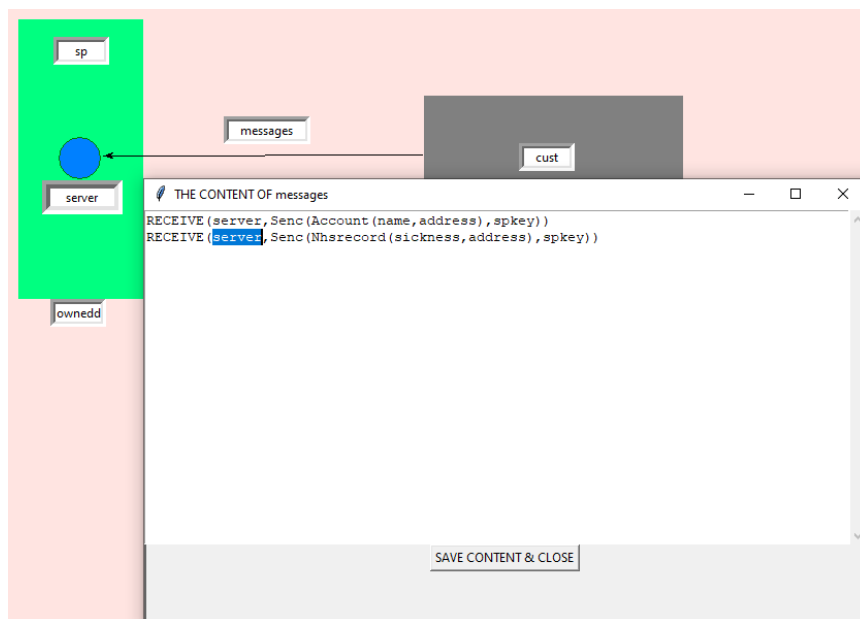
See “Examples-GUI/Example-Crypto2” in the “Pol and arch files used in the manual v.0.9.8.zip”.

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.



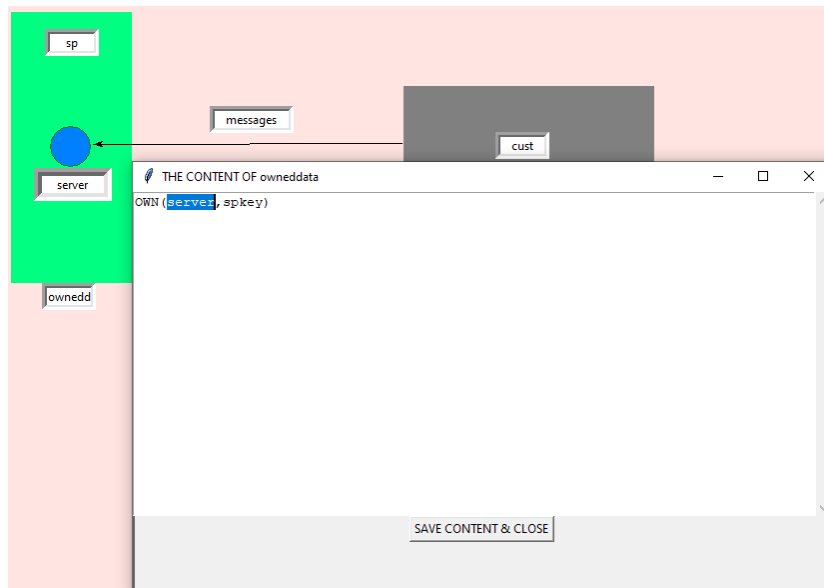
Then, in *messages*, we specify two RECEIVE actions, saying that **server** (instead of *sp*) can receive an encrypted account that contains a name and an address inside it. The encryption is symmetric encryption using the shared key *spkey*.

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



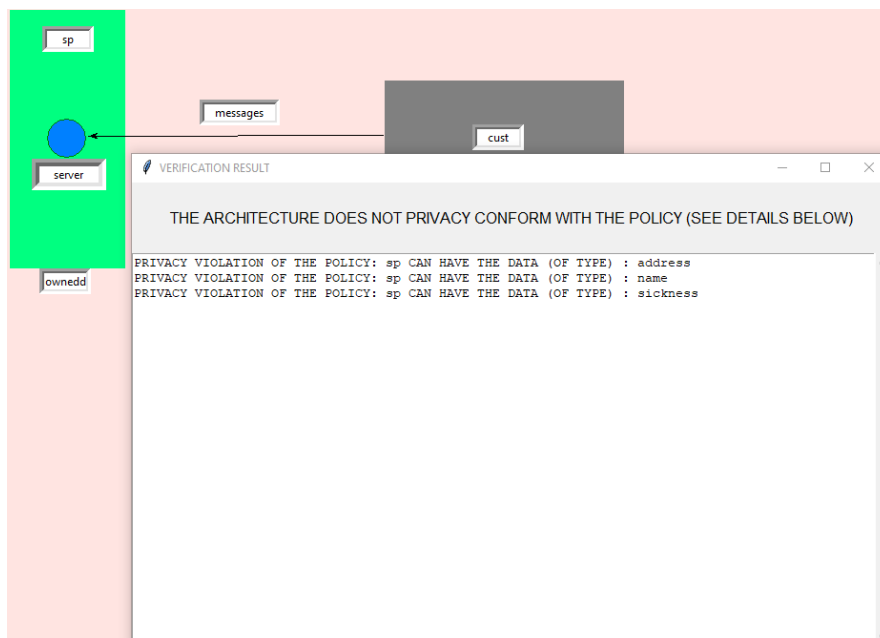
In *owneddata*, we specify an action **OWN** for *server*:

OWN(server,spkey)



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

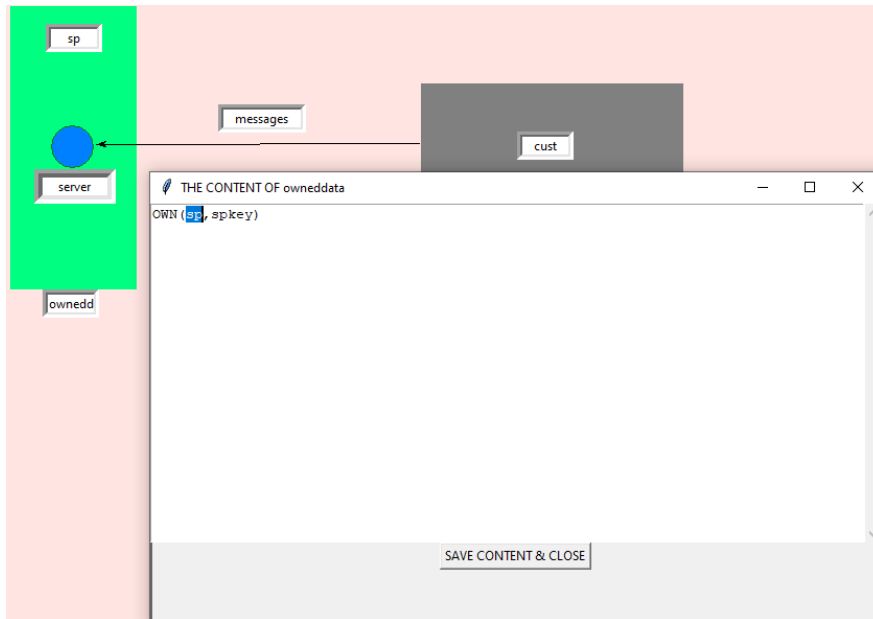
After that, the verification result is the same as in example 1, as the data that server handles is available to sp.



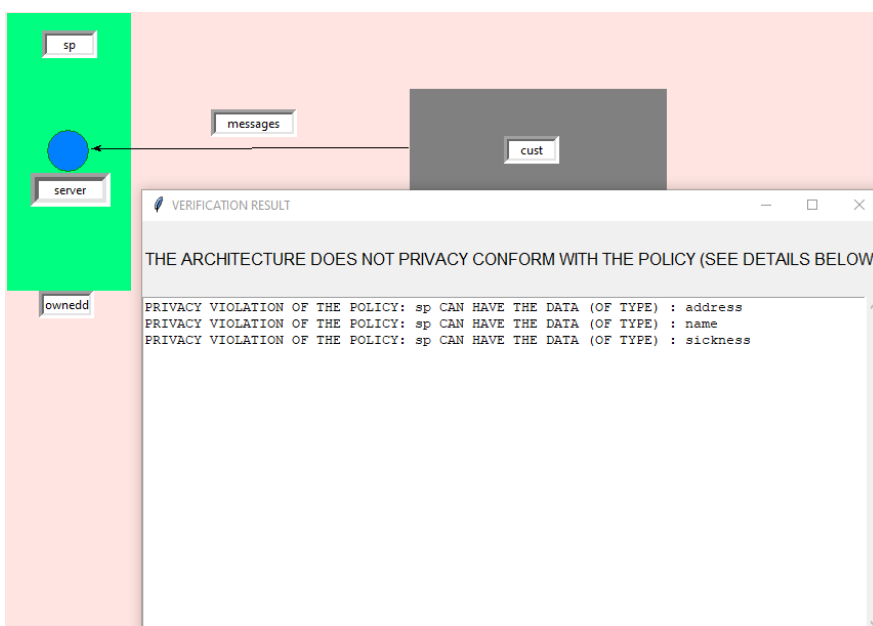
Example 3

See “Examples-GUI/Example-Crypto3” in the “Pol and arch files used in the manual v.0.9.8.zip”.

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

See “Examples-GUI/Example-Crypto4” in the “Pol and arch files used in the manual v.0.9.8.zip”.

Data connection sub-policy: Based on the data groups defined in the 1st example, in this example, we consider the verification whether an entity is to be able to link/connect two pieces of data of type name and sickness.

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 name

sickness

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

No

name-sickness:Any Link is Forbidden

ADD DATA CONNECTION POLICY

DELETE CONNECTION POLICY

CLOSE & SAVE

Choose an entity sp (service provider) Choose a data group name Choose a data type SPECIFY EACH

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.

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 address

sickness

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

No

address-sickness:Any Link is Forbidden

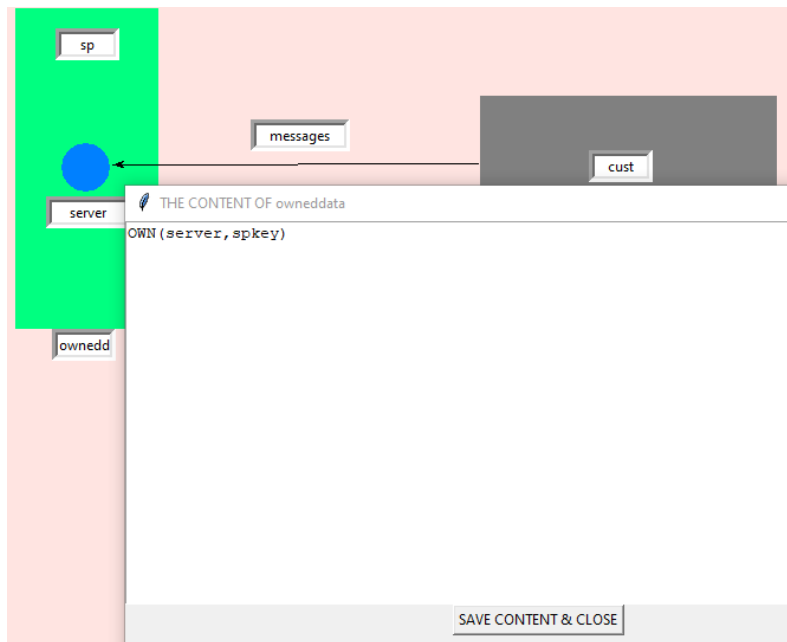
ADD DATA CONNECTION POLICY

DELETE CONNECTION POLICY

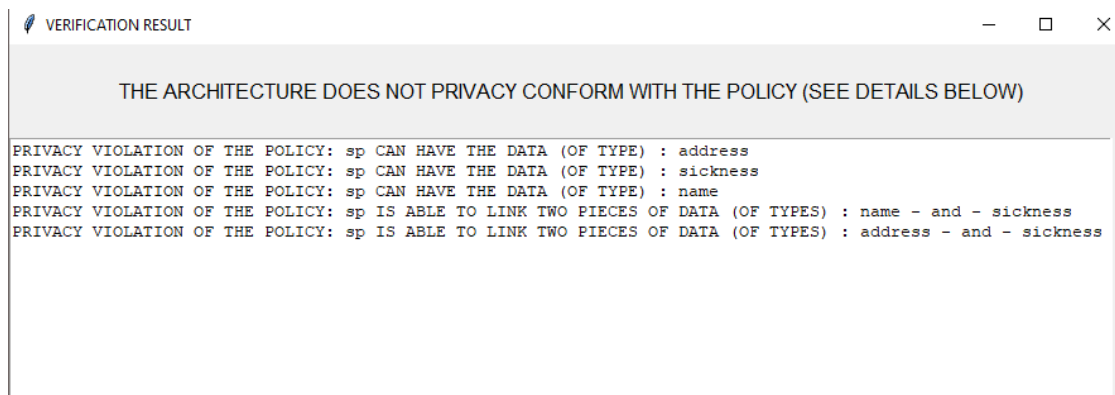
CLOSE & SAVE

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

See "Examples-GUI/Example-Crypto5" in the "Pol and arch files used in the manual v.0.9.8.zip".

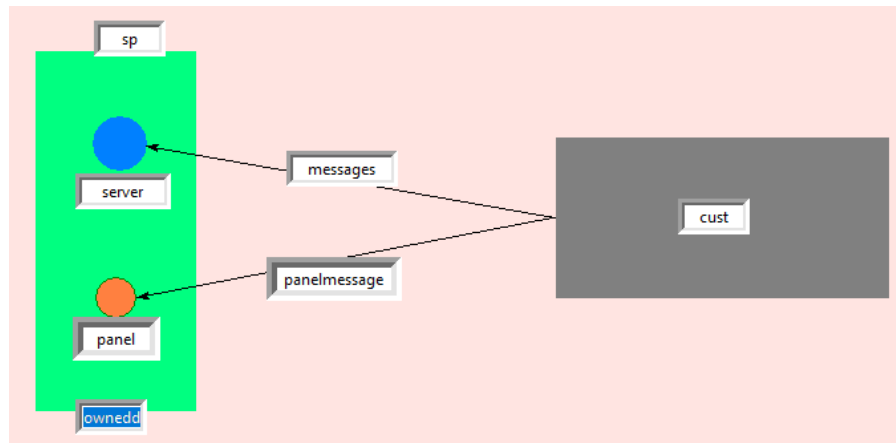
Now, besides server, we add an additional sub-component, called panel.

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.

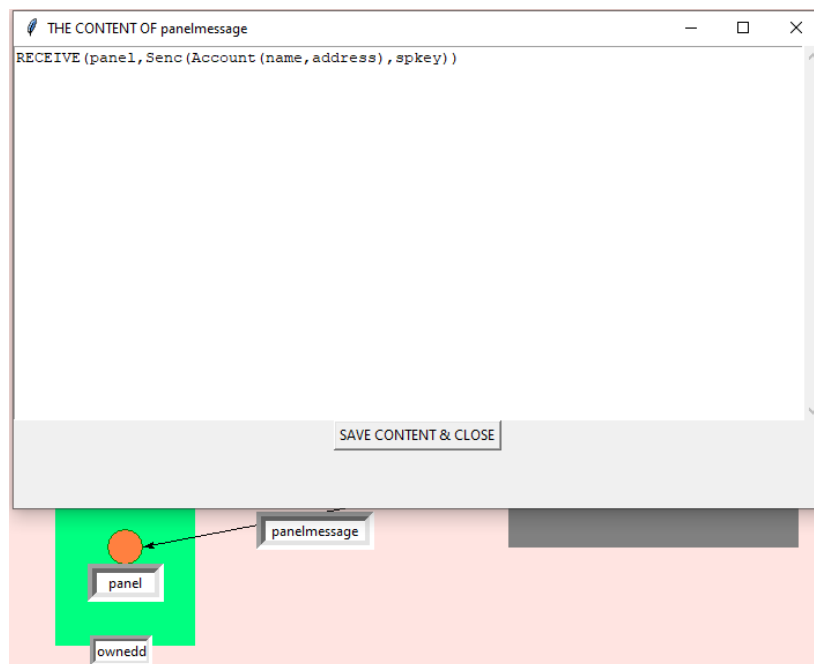
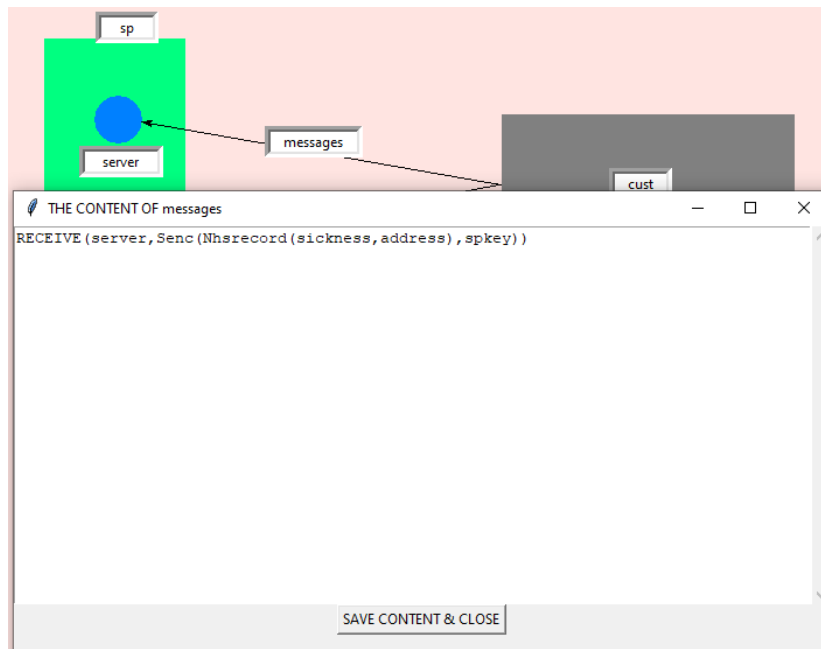
Also, the service provider is not allowed to be able to link *name with sickness* (in any form), and link address with sickness (in any form).

Architecture:



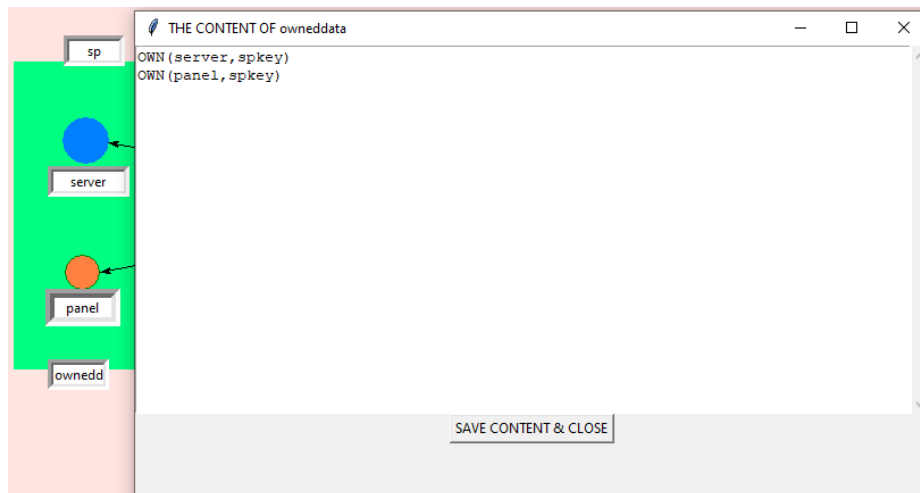
We add the following actions: the server receives a NHS (national health service) record that contains the information about the sickness and address, encrypted with the type of service provider key (shared key), spkey.

- **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.

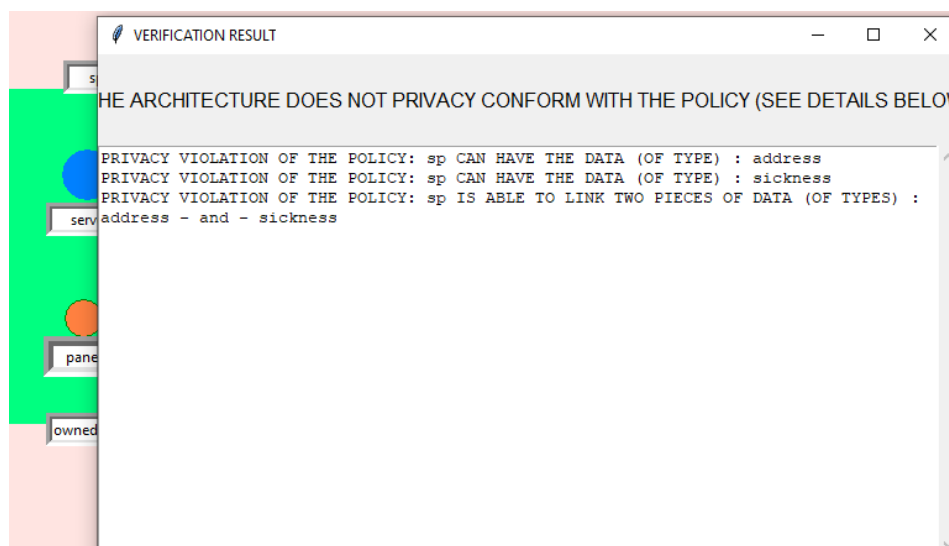
- **OWN**(panel,spkey)
- **OWN**(server,spkey)



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 of *name with sickness*, which violates the policy.

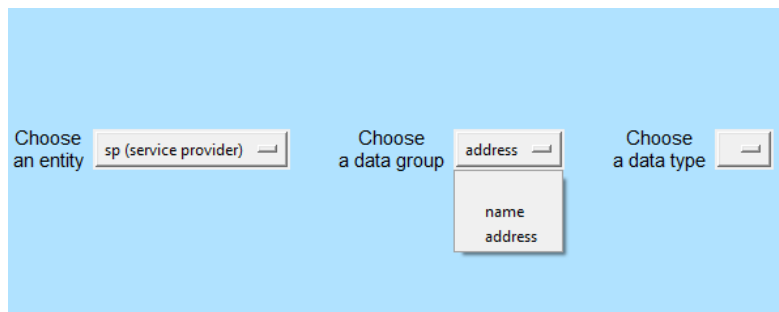
Example 6

See “Examples-GUI/Example-Crypto6” in the “**Pol and arch files used in the manual v.0.9.8.zip**”.

In this example, we introduce 2-layer nested crypto functions inside the datatypes.

Policy:

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



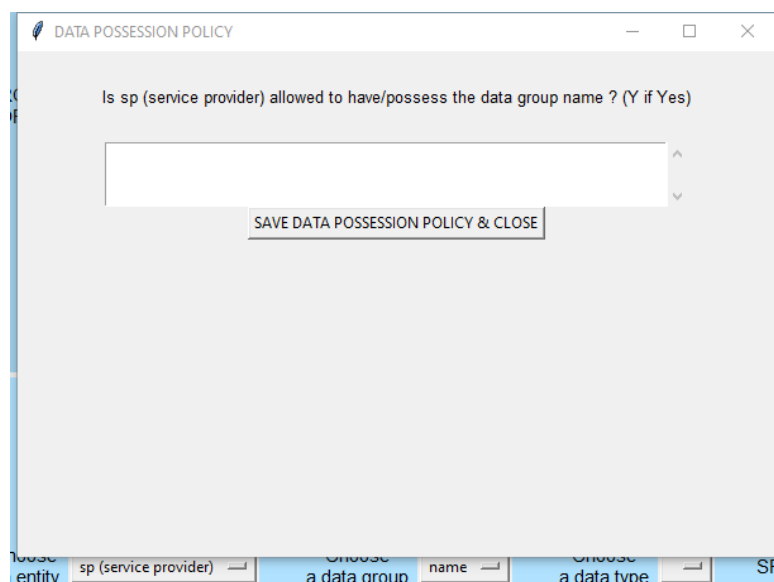
Choose an entity

Choose a data group

Choose a data type

name
address

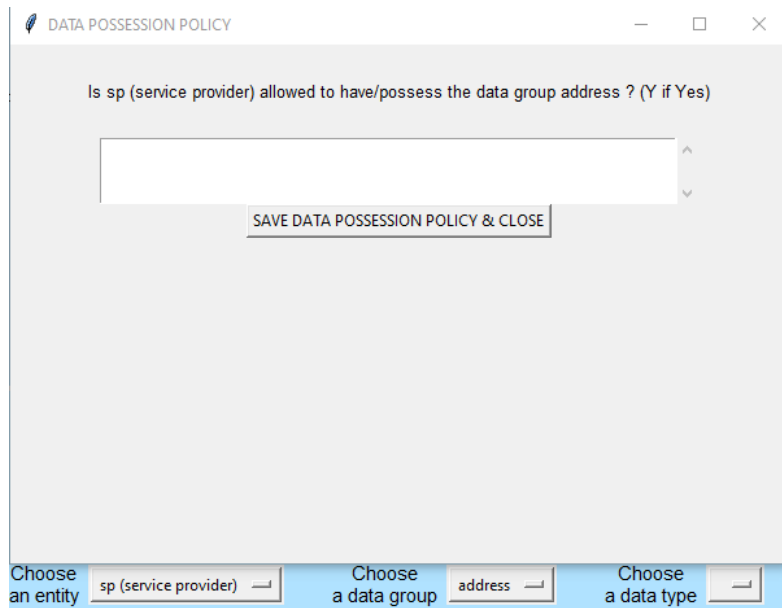
We do not allow the service provider to have a piece of data of type *name*, and *address*, as empty policy field means "NO" (FORBID).



DATA POSSESSION POLICY

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

SAVE DATA POSSESSION POLICY & CLOSE



Architecture:

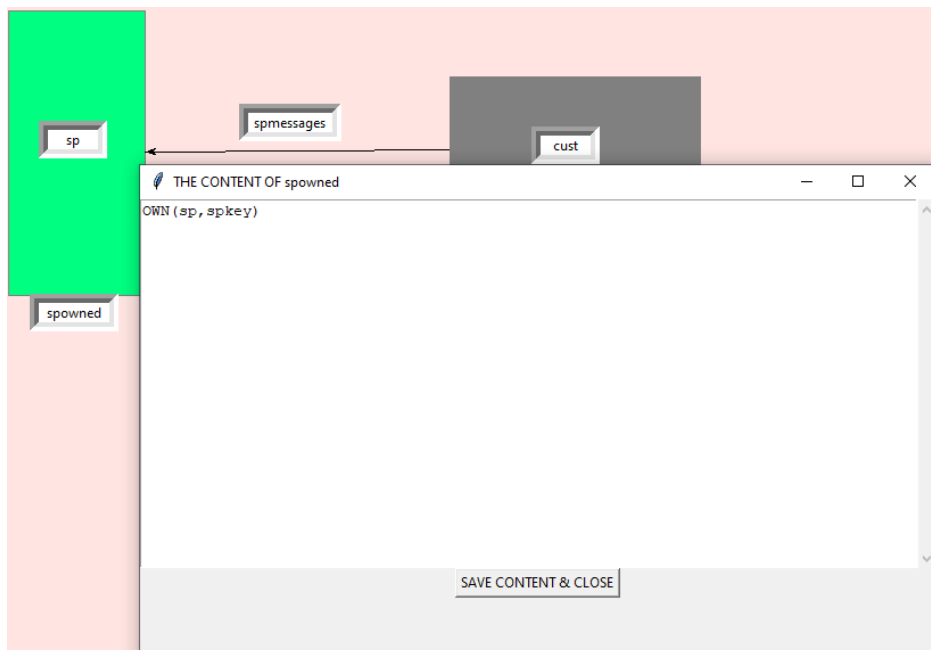
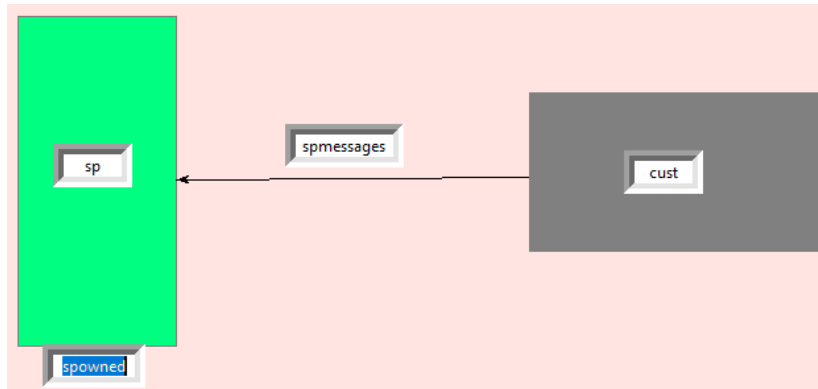
In the architecture we specify with actions that the service provider can receive an account that contains encrypted name (using the shared key spkey), and a cleartext address. The whole account is then again encrypted with the same key, spkey.

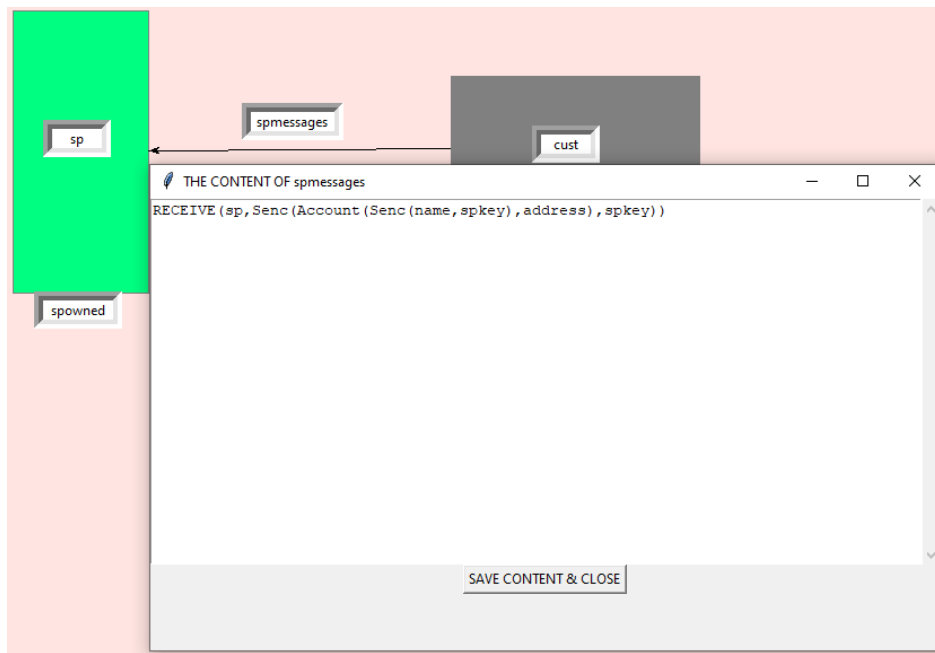
In the architecture level, we abstract away from how the key is shared and what concrete encryption algorithm is used. Those details are left to the protocol or implementation level.

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

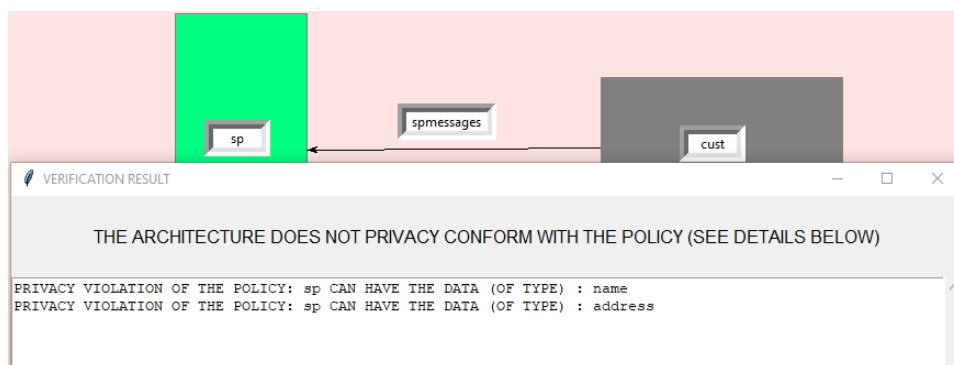
We also specify that the service provider owns spkey.

OWN(sp,spkey)





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



Example 7

See "Examples-GUI/Example-Crypto7" in the "Pol and arch files used in the manual v.0.9.8.zip".

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.

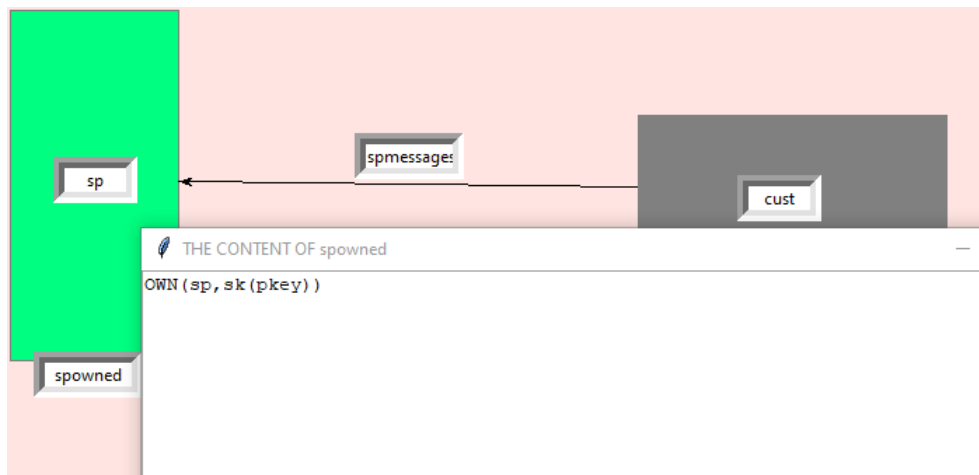


Now, spkey is encrypted using an asymmetric encryption func. (Aenc) with the public key pkey.

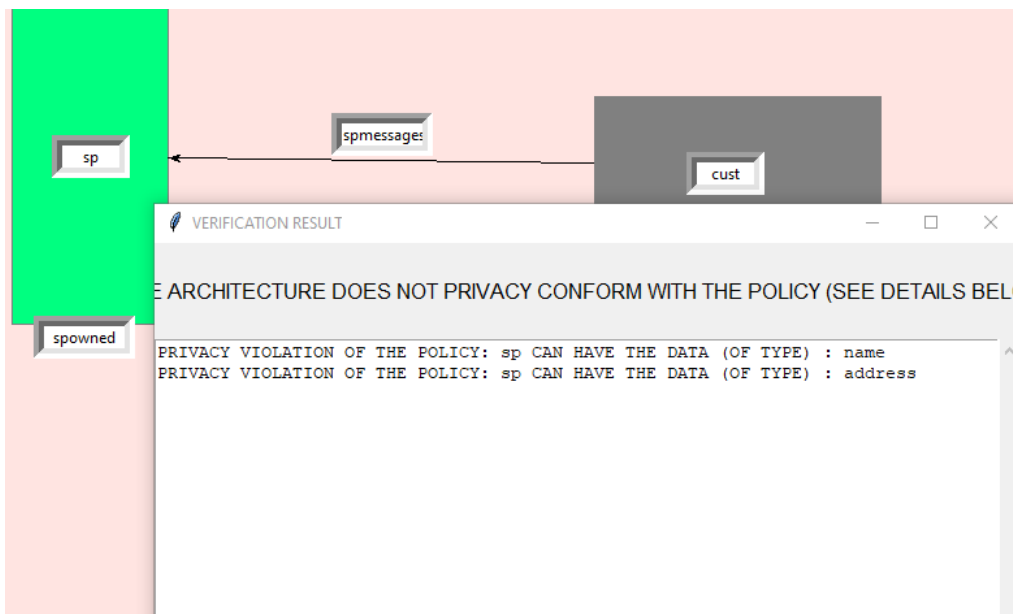
Finally, the service provider owns the secret/private key counter part of the public key pkey (namely, sk(pkey)).

- **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

See "Examples-GUI/Consent-Example-8a" in the "Pol and arch files used in the manual v.0.9.8.zip".

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.)

DATA COLLECTION POLICY

Does sp (service provider) collect consent before collecting the data (of type) Profile(photo,job) ? (give Y or N)

Y

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

create:account

SAVE COLLECTION POLICY & CLOSE

Choose a service provider: sp (service provider) | Choose a data group: Profile(photo,job) | Choose a data type: | SPECIFY

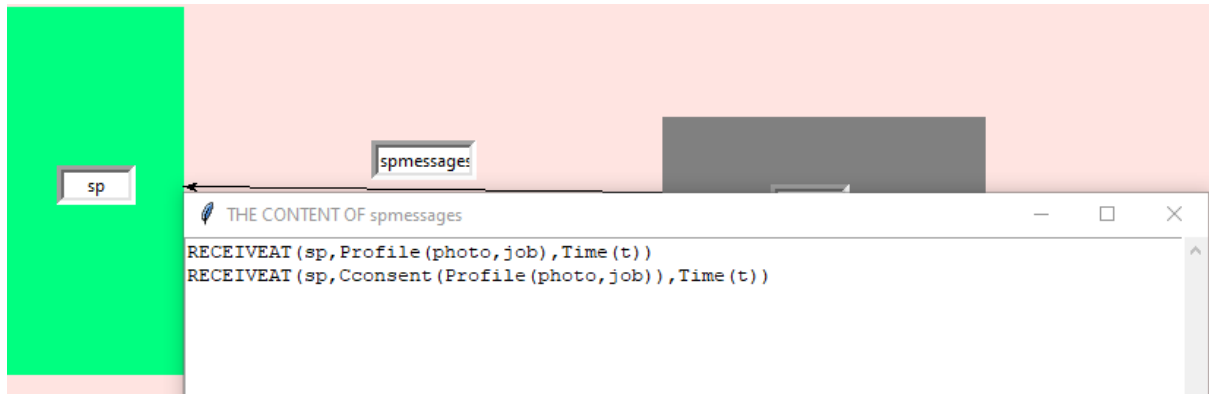
Architecture:

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))

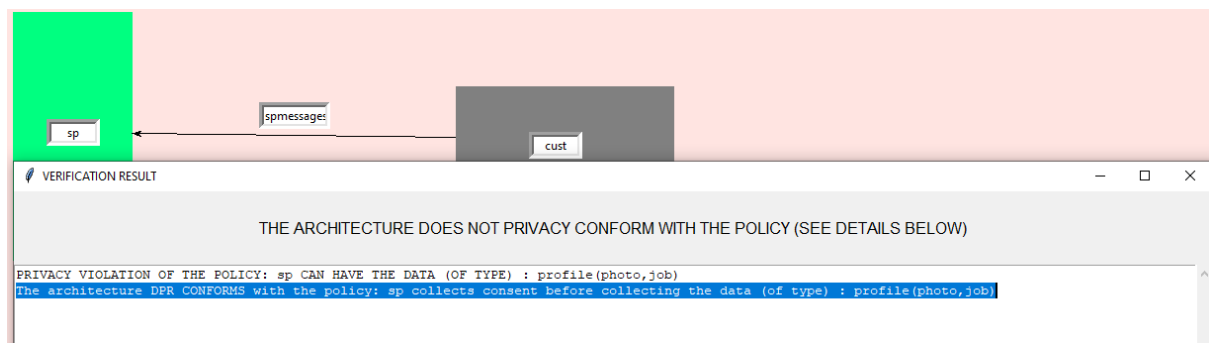
The service provider receives the corresponding collection consent on the data at the same time or before, this is abstractly modelled by the same time t.

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

See "Examples-GUI/Consent-Example-8b" in the "Pol and arch files used in the manual v.0.9.8.zip".

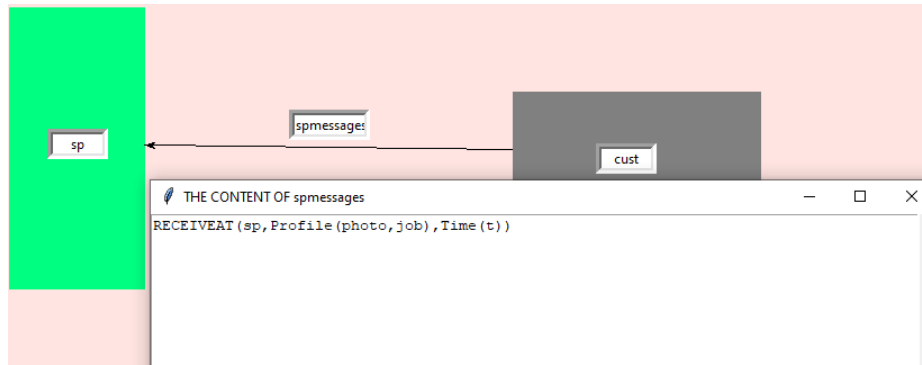
DPR VIOLATION CASE:

The policy is unchanged compared to the

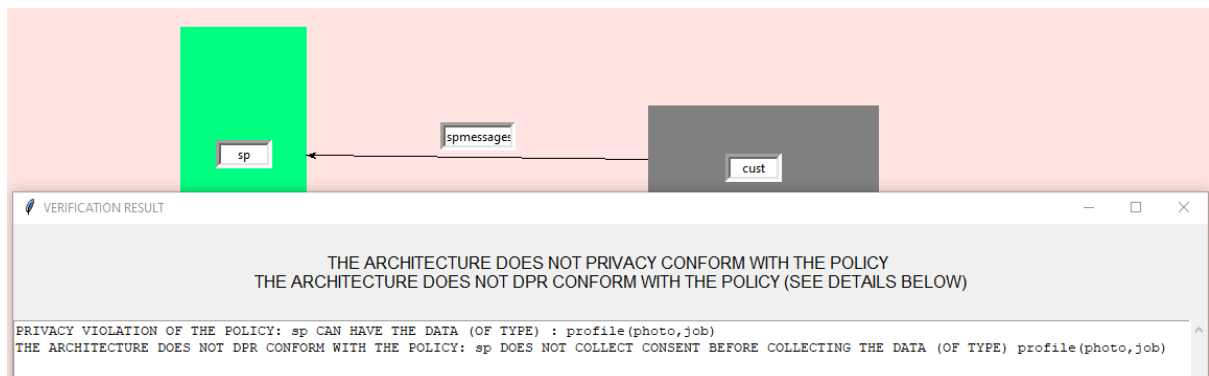
Architecture:

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.

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



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



Example 8/c

See "Examples-GUI/Consent-Example-8c" in the "[Pol and arch files used in the manual v.0.9.8.zip](#)"

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".

DATA COLLECTION POLICY

Does sp (service provider) collect consent before collecting the data (of type) Profile(photo,job) ? (give Y or N)

Y

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

create:account

SAVE COLLECTION POLICY & CLOSE

Choose a data group: Profile(photo,job) | Choose a data type: SPECIFY

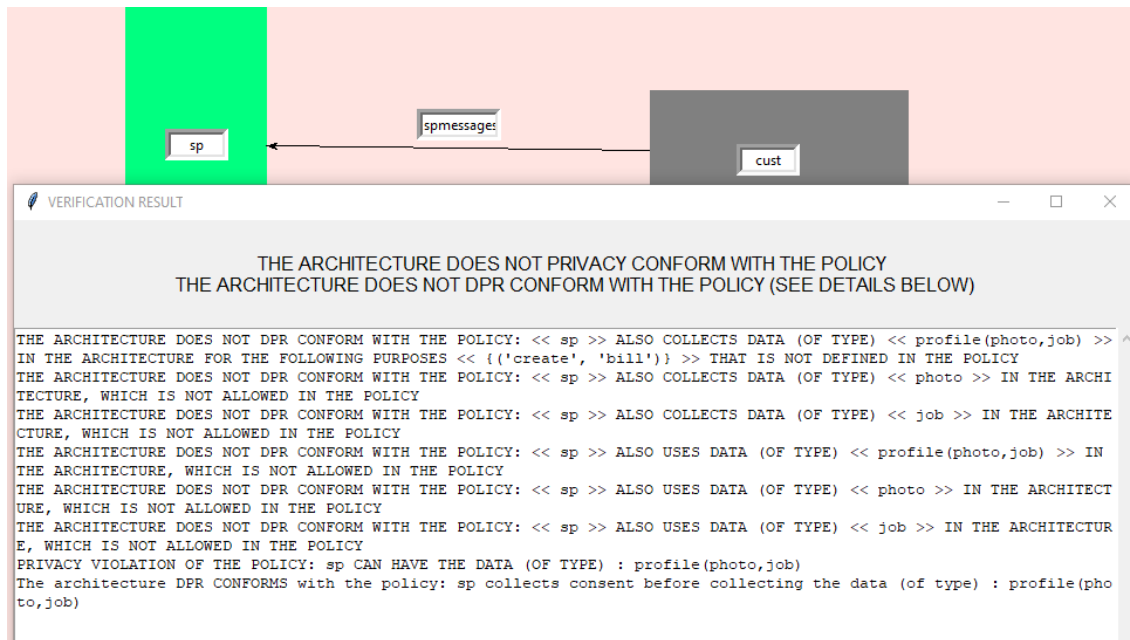
Architecture: In the architecture, we define an additional action CREATE that says that sp creates a bill based on a profile that contains a photo and a piece of job information.

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



Verification results:

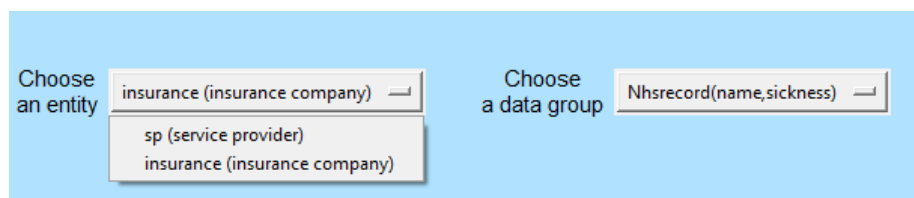
As we can see, results show that there are DPR violations because (in the 1st line) the profile can be collected in the architecture for the purpose of creating a bill, but this is not specified/allowed in the policy. The next lines also DPR violation because we forgot to specify the collection for the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



Example 9/a

See “Examples-GUI/Transfer-Example-9a” in the “**Pol and arch files used in the manual v.0.9.8.zip**”.

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)

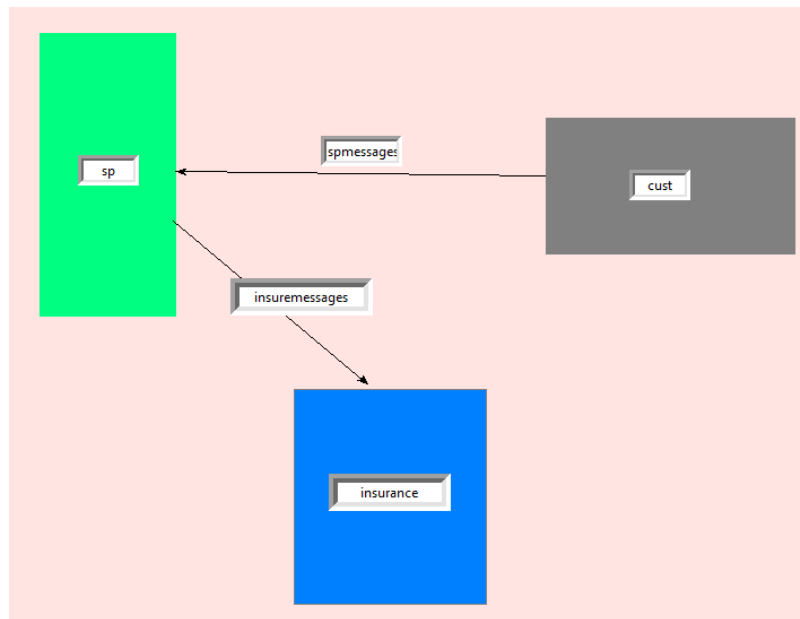
SAVE TRANSFER POLICY & CLOSE

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

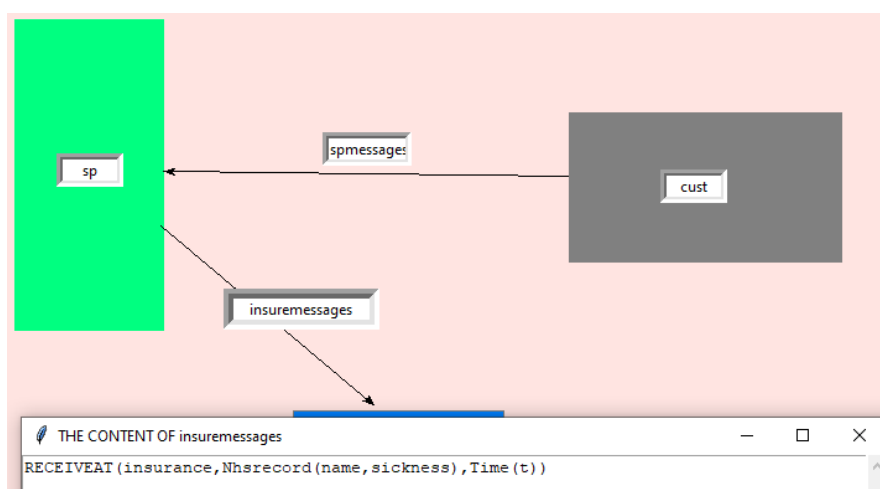
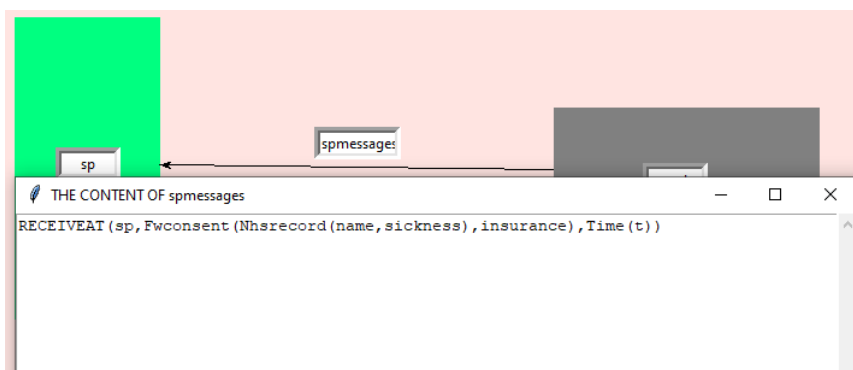
Architecture:

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)**)



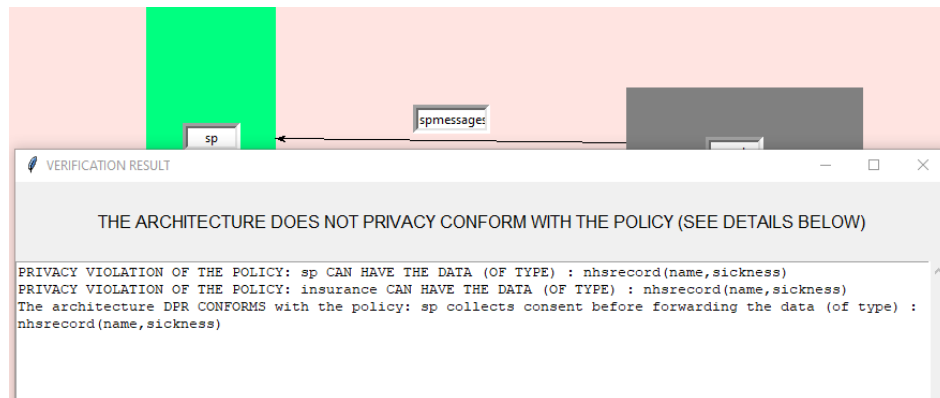
We specify the two messages can be received by sp and insurance:



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

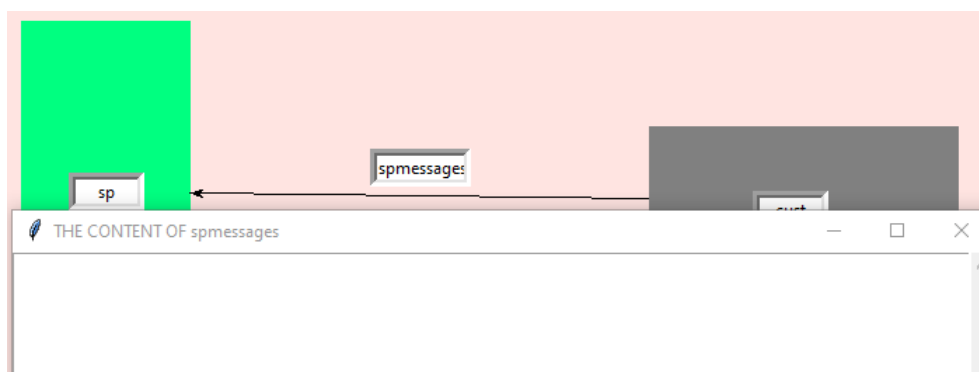
See "Examples-GUI/Transfer-Example-9b" in the "Pol and arch files used in the manual v.0.9.8.zip".

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

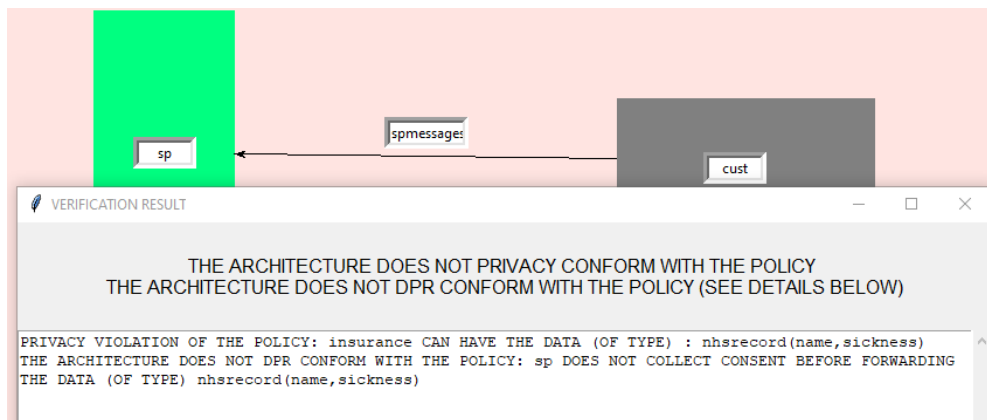
RECEIVEAT(insurance,Nhsrecord(name,sickness),Time(t))

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.



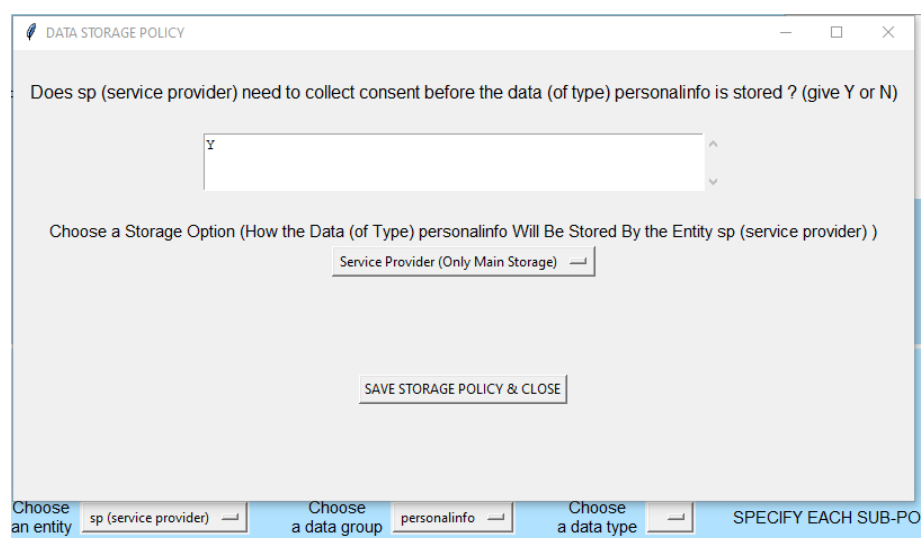
Example 10

See "Examples-GUI/Storage-Example-10" in the "Pol and arch files used in the manual v.0.9.8.zip".

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.

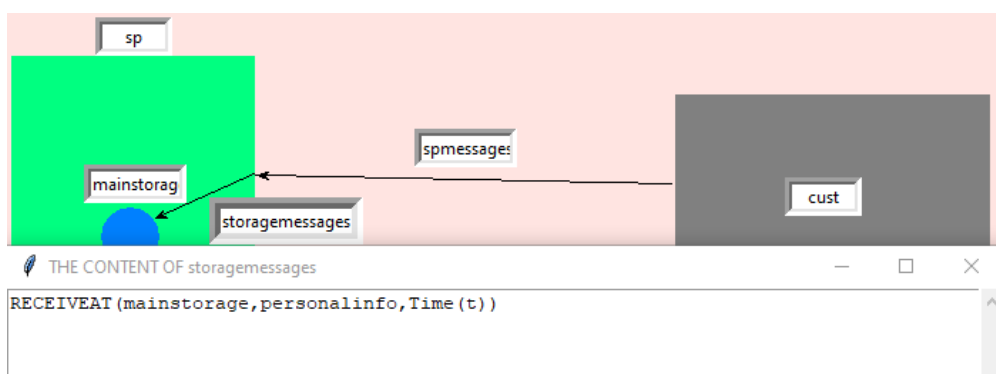
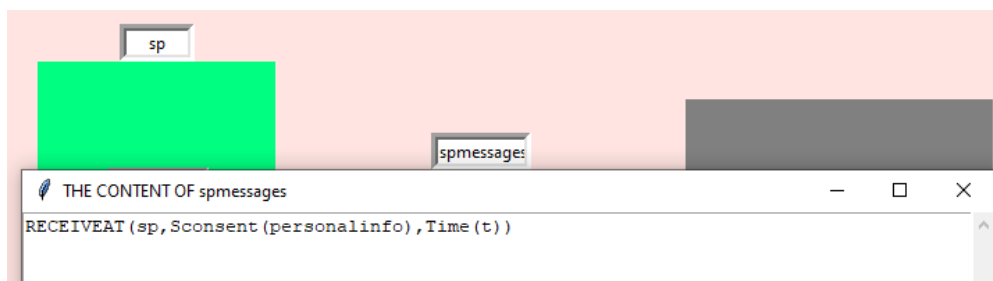
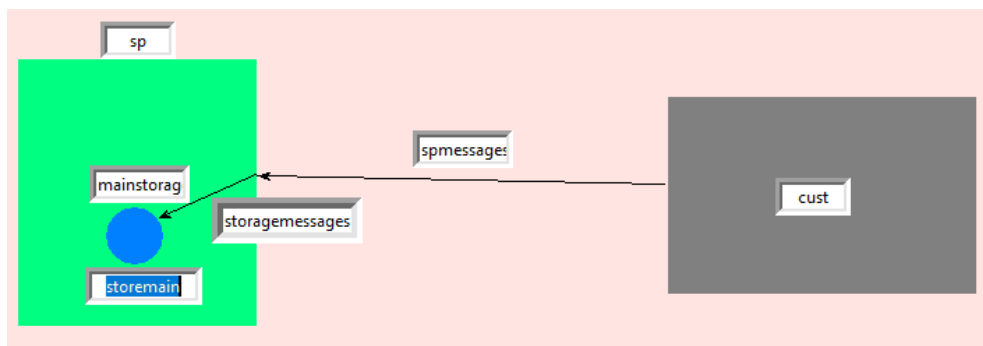
Again, we leave the data possession policy bank, which forbid for sp to be able to have the data of type personalinfo. We also, leave the collection policy and usage policy blank.

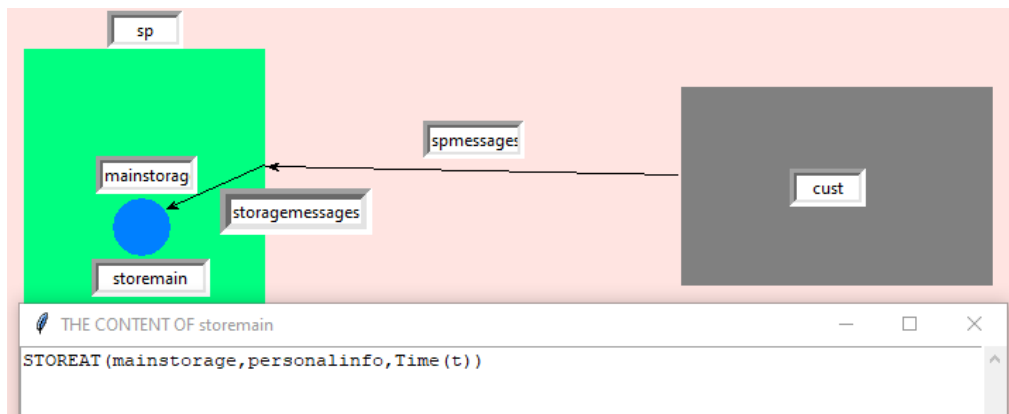


Architecture:

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`))





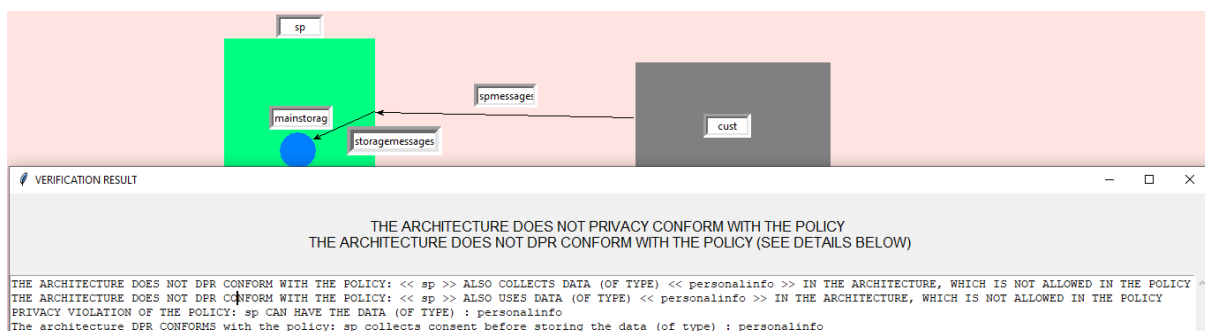
We got the not well-formed architecture message because the architecture is not well-formed, mainstorage store the data of type personalinfo, but has not received it before/or own it/created/calculated it before.

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).



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

Example 11

See “Examples-GUI/Deletion-Example-11” in the “**Pol and arch files used in the manual v.0.9.8.zip**”.

Deletion sub-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 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 RETENTION POLICY

Only From Main 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)

8y

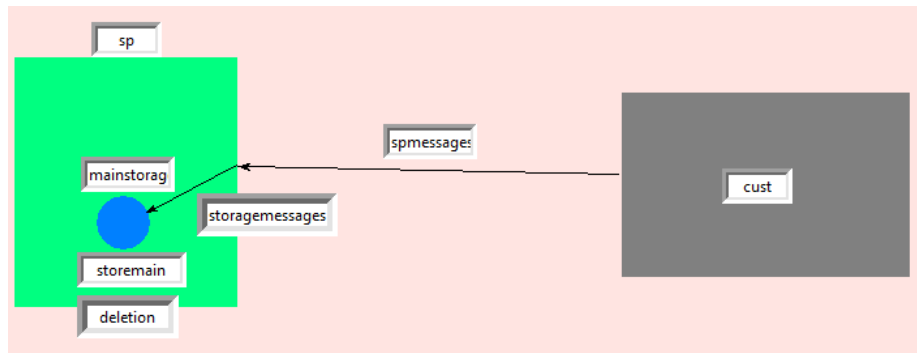
SAVE DELETION POLICY & CLOSE

Choose an entity sp (service provider) Choose a data group personalinfo Choose a data type SPECIFY

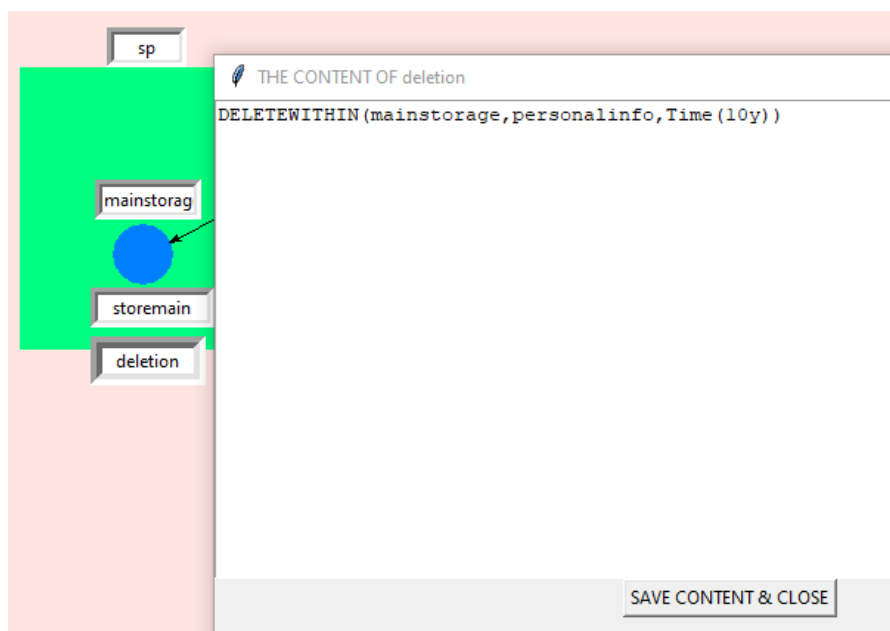
Architecture:

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.

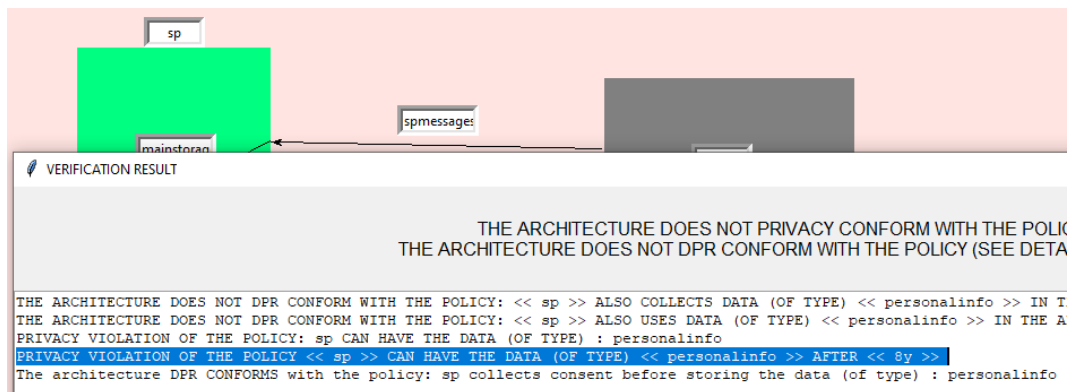
- **RECEIVEAT**(sp,Sconsent(personalinfo),**Time**(t))
- **STOREAT**(mainstorage,personalinfo,**Time**(t))
- **RECEIVEAT**(mainstorage,personalinfo,**Time**(t))
- **DELETEWITHIN**(mainstorage,personalinfo,**Time**(10y)).



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

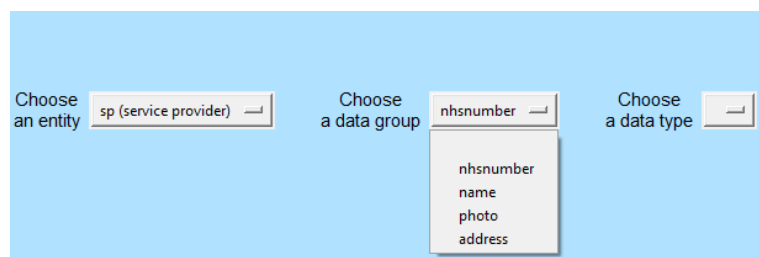
See “Examples-GUI/Meta-Example-12a” in the “Pol and arch files used in the manual v.0.9.8.zip”.

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.



Then, we forbid (any kind of linkability, 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

address

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

No

nhsnumber-photo:Any Link is Forbidden

ADD DATA CONNECTION POLICY

DELETE CONNECTION POLICY

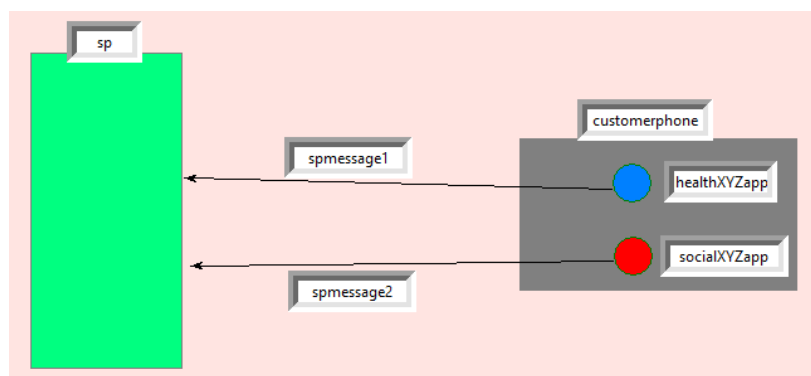
CLOSE & SAVE

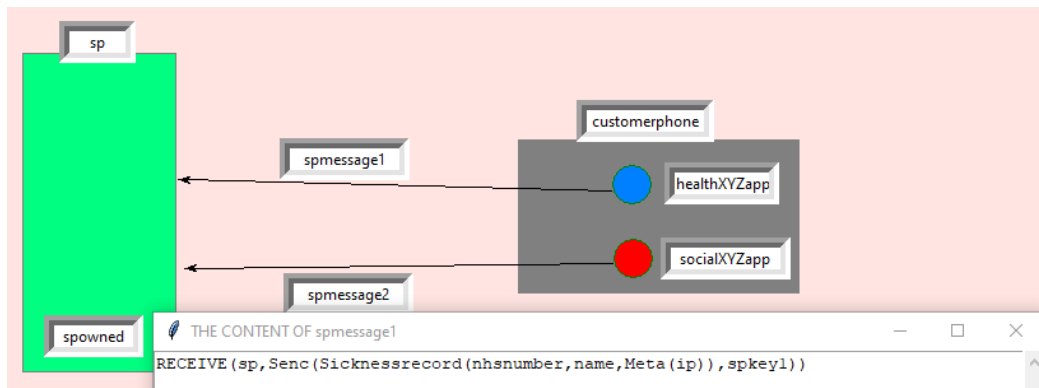
Choose an entity sp (service provider) Choose a data group nhsnumber Choose a data type SPECIFY

Architecture:

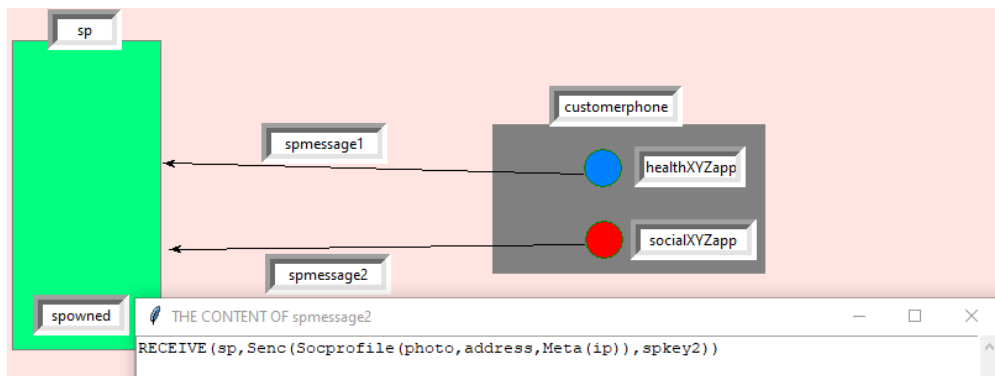
In the architecture, a service provider collects data from two phone applications. The “HealthXYZ” app sends the service provider a sickness record with an ip address (phone ip) other app, called, “SocialXYZ” also sends the social profile with the same ip address (same phone).

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

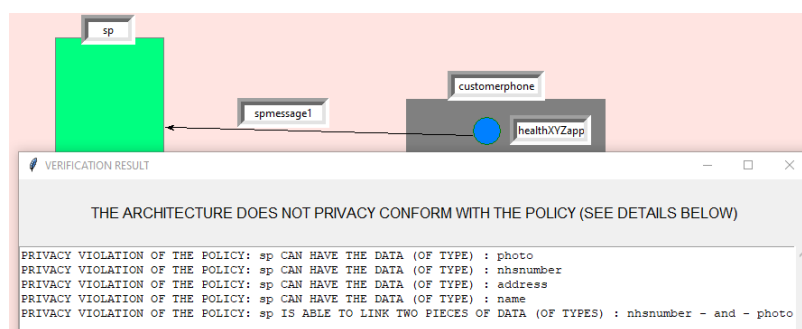




Here we define the two messages *spmessages1* and *spmessages2*.



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

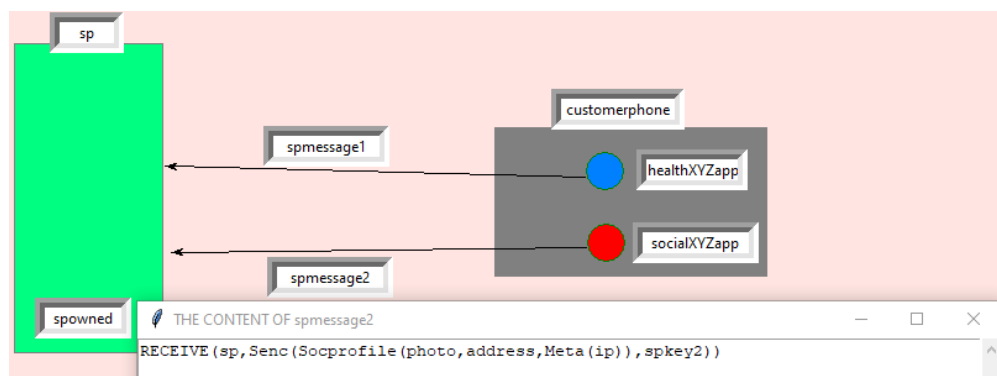
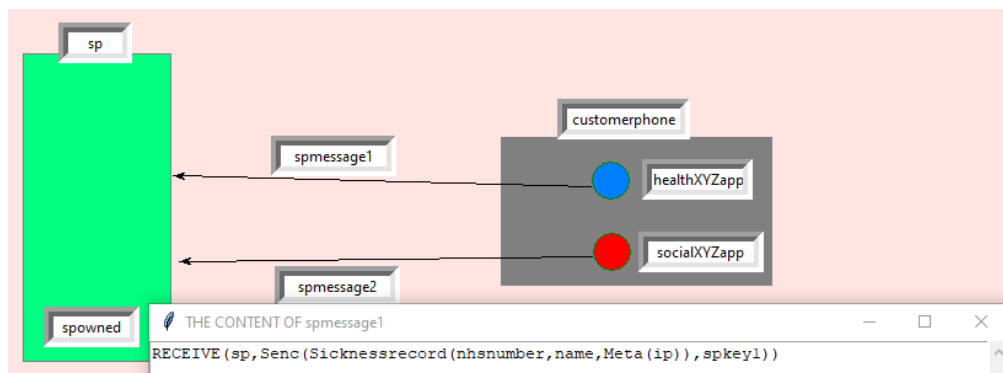
See “Examples-GUI/Meta-Example-12b-Enc/arch12encv1.arch” in the “**Pol and arch files used in the manual v.0.9.8.zip**”.

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

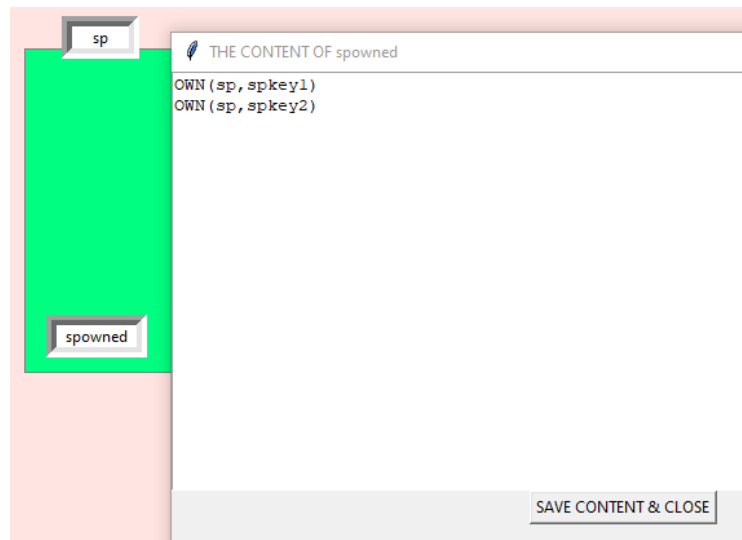
Architecture: The service provider, *sp*, receives the same data but with symmetric encryption, and it owns the decryption key for both messages.

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

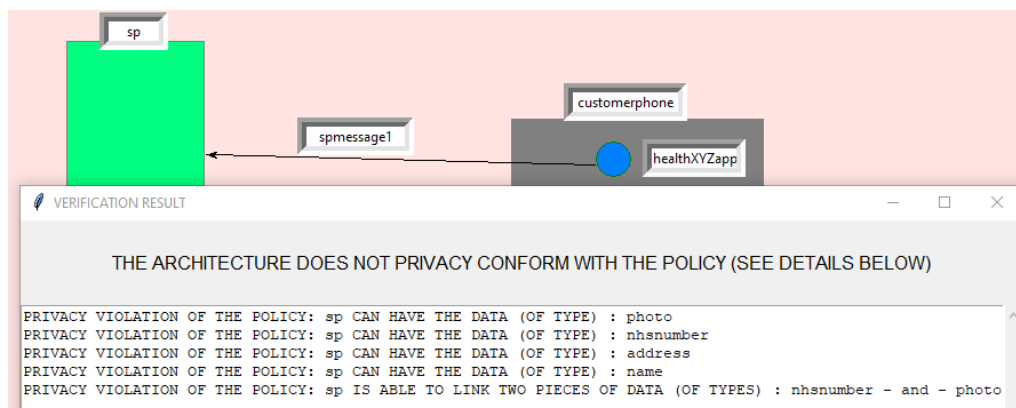
The messages are unchanged compared to the previous example:



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.



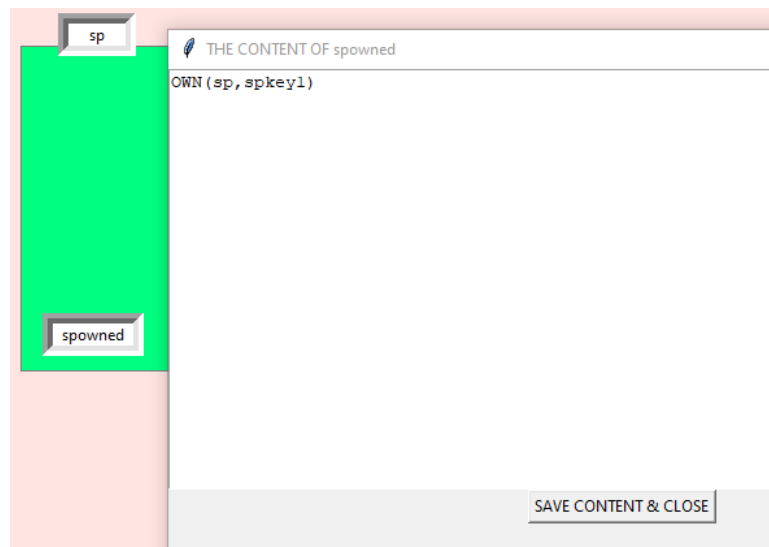
Example 12/c

See “Examples-GUI/Meta-Example-12b-Enc/arch12encv2.arch” in the “[Pol and arch files used in the manual v.0.9.8.zip](#)”.

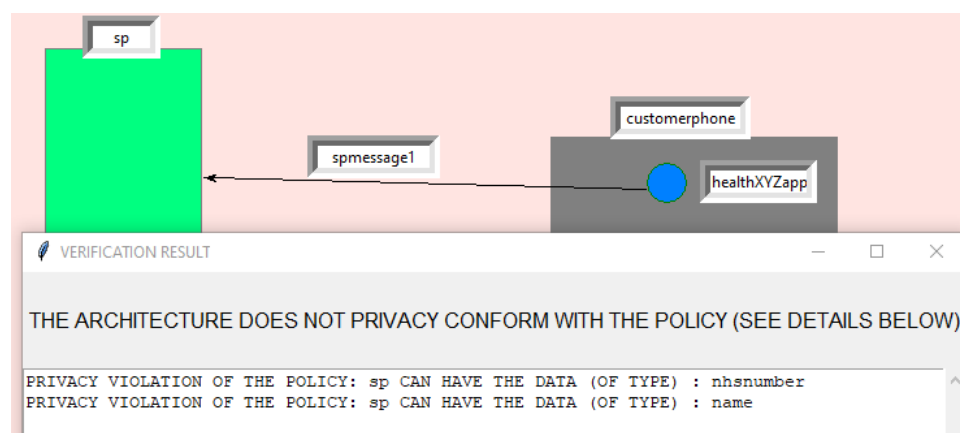
Here, the policy is the same as in the point 12/a.

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)

See “Pseudonym – Example-13” in the “[Pol and arch files used in the manual v.0.9.8.zip](#)”.

In the version v0.9.8, 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

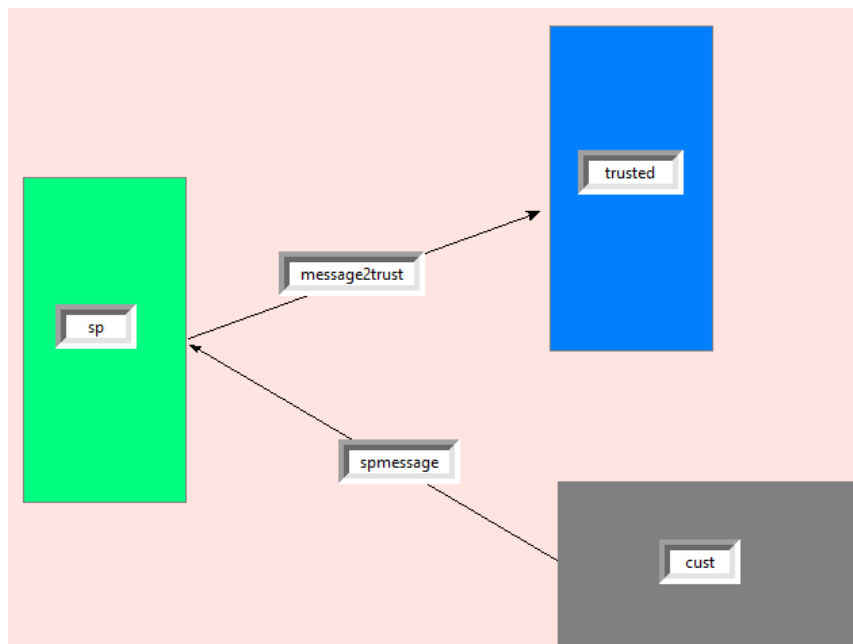
DATA POSSESSION POLICY

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

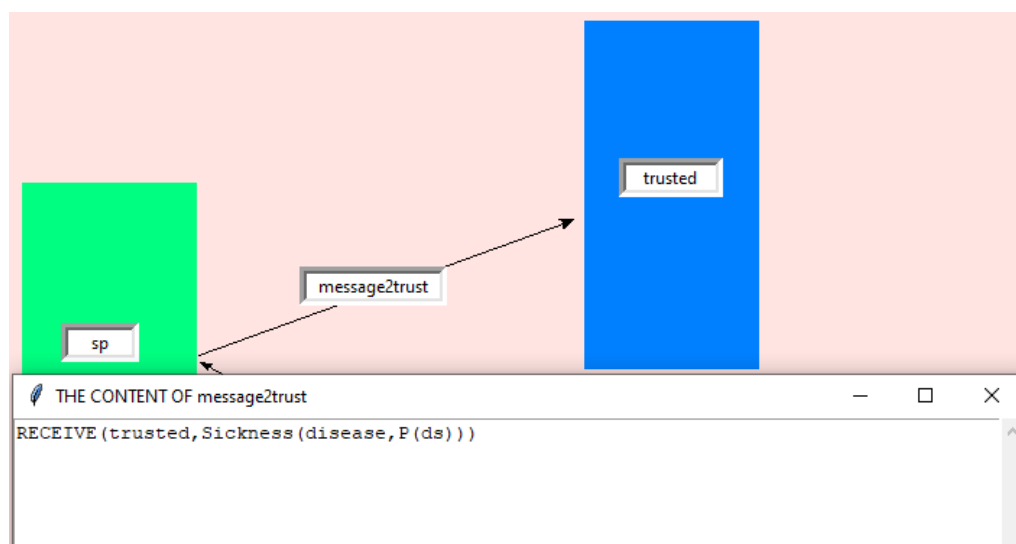
SAVE DATA POSSESSION POLICY & CLOSE

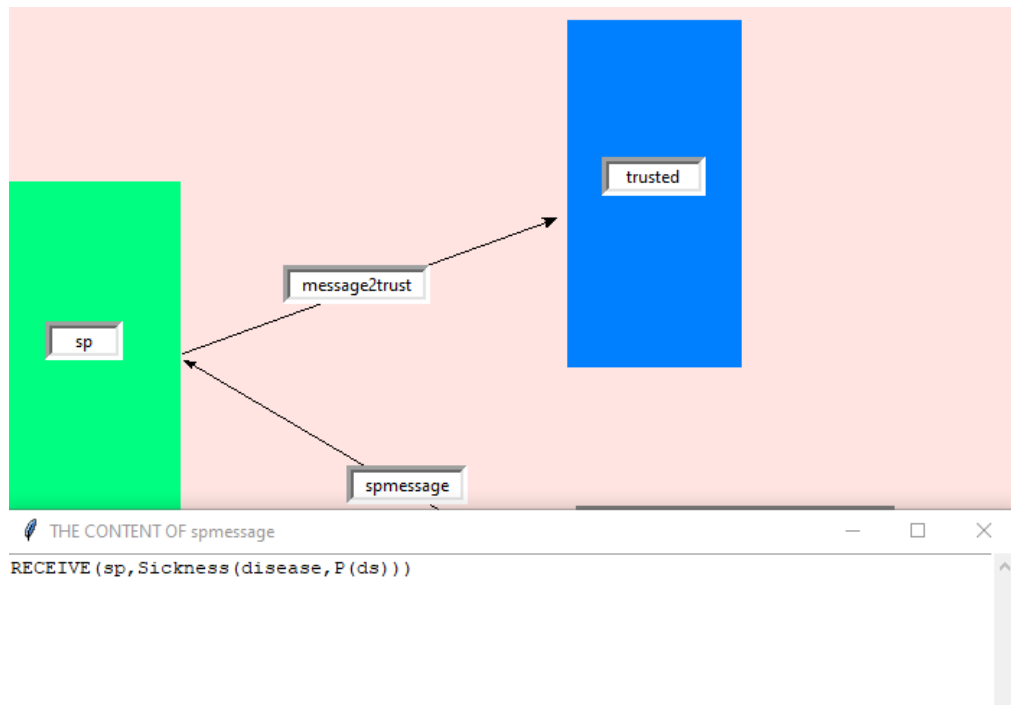
Choose an entity sp (service provider) Choose a data group Sickness(disease,ds) Choose a data type

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)**.





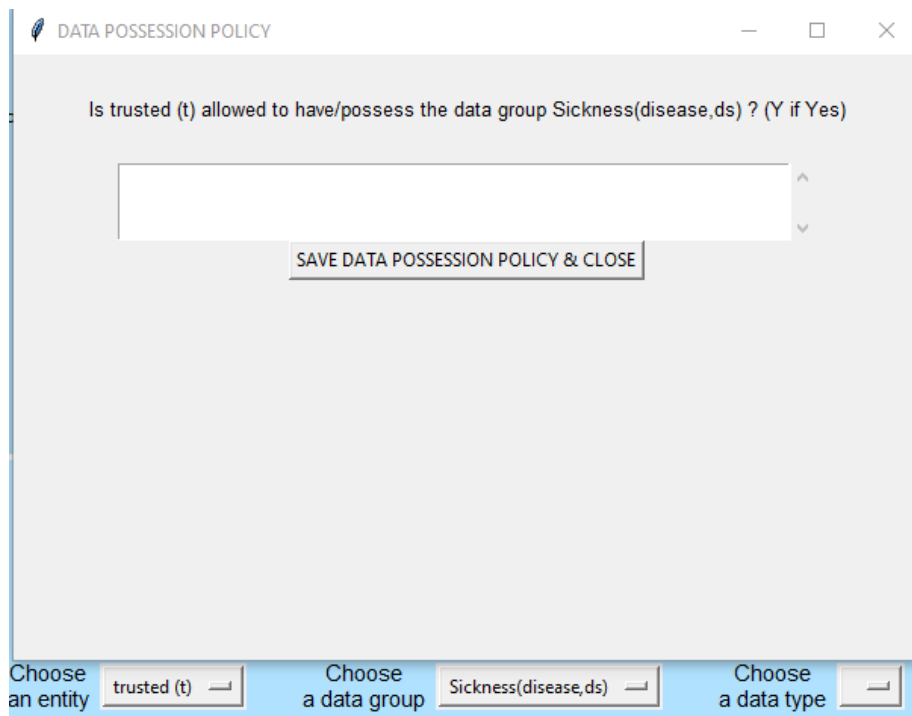
Verification result (conformance):

VERIFICATION RESULT

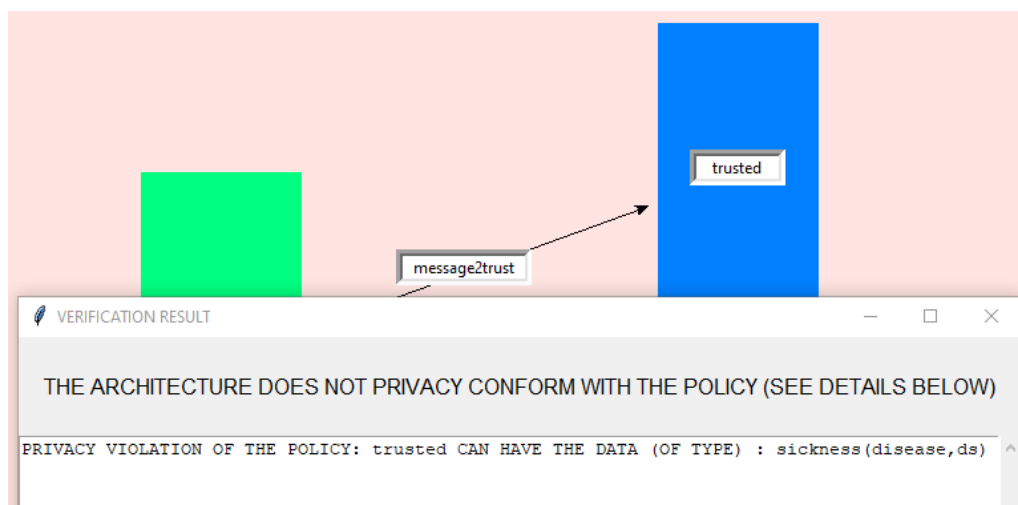
THE ARCHITECTURE PRIVACY CONFORMS WITH THE POLICY
THE ARCHITECTURE FUNCTIONALLY CONFORMS WITH THE POLICY
THE ARCHITECTURE DPR CONFORMS WITH THE POLICY (SEE DETAILS BELOW)

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)*.



Verification result (privacy violation)



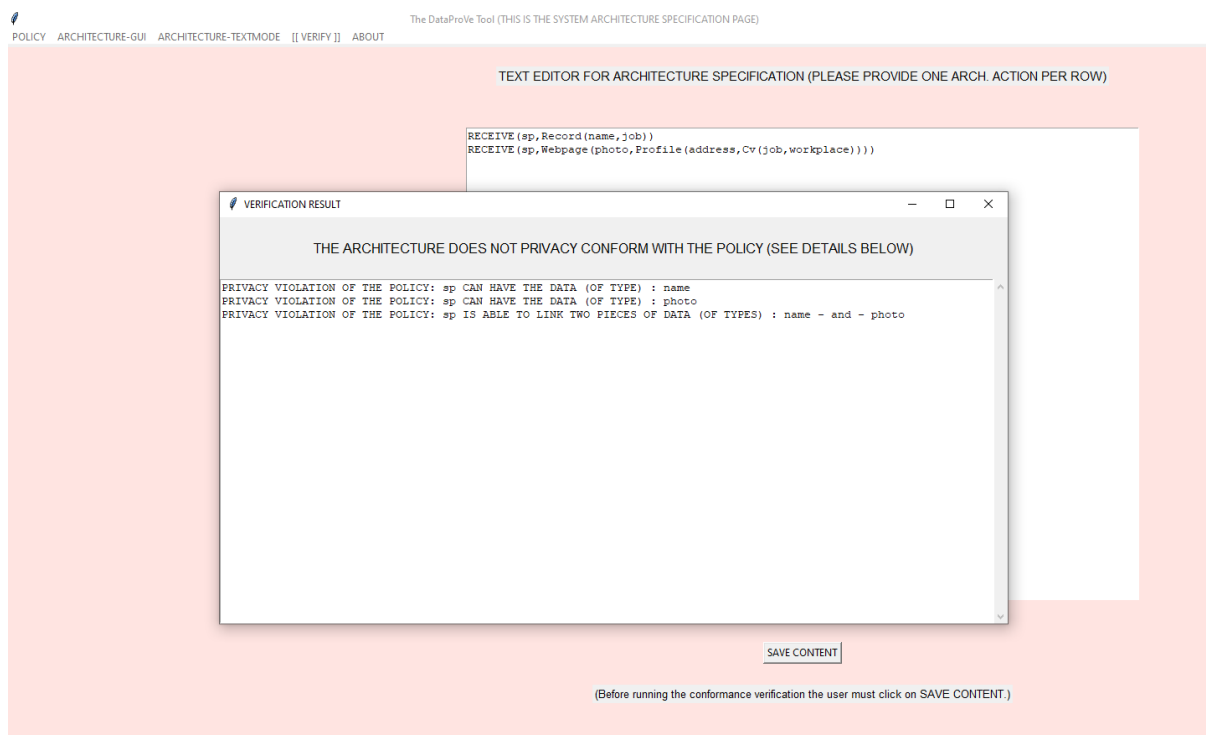
TEXT-MODE EXAMPLES

Example 1 (LINK)

("Examples-TEXTMODES/ Examples-TEXTMODE-Link/arch1-textmode.txt" in "Pol and arch files used in the manual v.0.9.8.zip")

In the examples 1-4, we check whether a piece of data of type name and photo can be linked by the service provider. In each case, we use text mode and specify different messages can be received by a service provider sp.

```
RECEIVE(sp,Record(name,job))
RECEIVE(sp,Webpage(photo,Profile(address,Cv(job,workplace))))
```



Example 2 ([LINK](#))

(“Examples-TEXTMODES/ Examples-TEXTMODE-Link/arch2-textmode.txt” in “Pol and arch files used in the manual v.0.9.8.zip”)

```
RECEIVE(sp,Record(name,job))
RECEIVE(sp,Webpage(Profile(address,Cv(photo,workplace)),job))
```

With the same verification result as above.

Example 3 ([LINK](#))

(“Examples-TEXTMODES/ Examples-TEXTMODE-Link/arch3-textmode.txt” in “Pol and arch files used in the manual v.0.9.8.zip”)

```
RECEIVE(sp,Record(name,job))  
RECEIVE(sp,Webpage(Profile(address,Cv(photo,workplace)),Insurance(number,job)))
```

With the same verification result as above.

Example 4 ([LINK](#))

(“Examples-TEXTMODES/ Examples-TEXTMODE-Link/arch4-textmode.txt” in “Pol and arch files used in the manual v.0.9.6.zip”)

```
RECEIVE(sp,Record(name,Profile(address,Cv(job,workplace))))  
RECEIVE(sp,Webpage(photo,job))
```

With the same verification result as above.

Example 5 ([LINKUNIQUE](#))

(“Examples-TEXTMODES/ Examples-TEXTMODE-LinkUnique/arch1-textmode-linkunique.txt” in “Pol and arch files used in the manual v.0.9.8.zip”)

In the examples 5-7, we check whether a piece of data of type *name* and *photo* can be UNIQUELY linked by the service provider. In each case, we use text mode and specify different messages can be received by a service provider sp.

In the policy, we set the data of type *nhsnumber* to be unique.

PROVIDE A NEW ENTITY:
 PROVIDE A DESCRIPTION:
 ADD NEW ENTITY

PROVIDE A GROUP OF DATA TYPES:
 IS THIS UNIQUE?
 THE DATA TYPES IN THIS GROUP:
 ADD DATA GROUP & TYPES

Choose an entity
 Choose a data group
 Choose a data type
 SUB-POLICIES :

Data Collection	Data Usage	Data Storage	Data Retention	Data Transfer	Data Possession	Data Connection (Permit)	Data Connection (Forbid)
-----------------	------------	--------------	----------------	---------------	-----------------	--------------------------	--------------------------

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 name

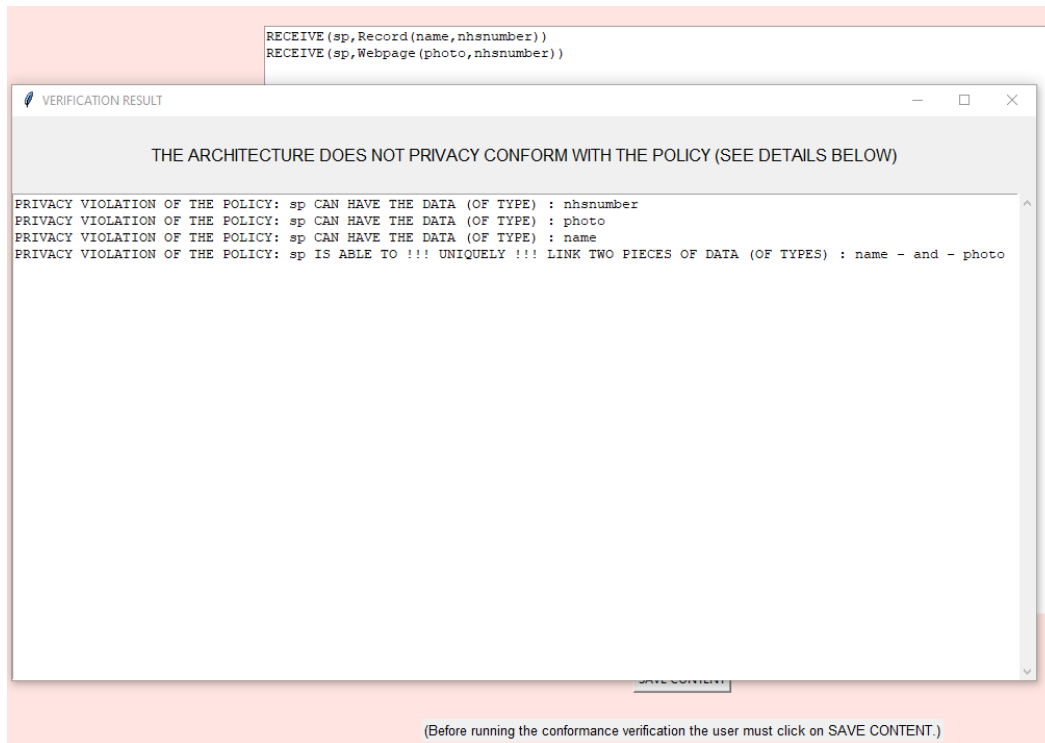
 Do you FORBID sp (service provider) only to uniquely link these two types of data ?

name-photo:Unique Link is Forbidden

ADD DATA CONNECTION POLICY
 DELETE CONNECTION POLICY
 CLOSE & SAVE

In the text mode, we define the architecture as

```
RECEIVE(sp,Record(name,nhsnumber))  
RECEIVE(sp,Webpage(photo,nhsnumber))
```



Example 6 (LINKUNIQUE)

("Examples-TEXTMODES/ Examples-TEXTMODE-LinkUnique/arch2-textmode-linkunique.txt" in "Pol and arch files used in the manual v.0.9.8.zip")

```
RECEIVE(sp,Heathinsurance(name,nhsnumber))  
RECEIVE(sp,Nhsrecord(photo,Profile(address,Bill(nhsnumber,date))))
```

With the same verification result as above.

Example 7 (LINKUNIQUE)

("Examples-TEXTMODES/ Examples-TEXTMODE-LinkUnique/arch3-textmode-linkunique.txt" in "Pol and arch files used in the manual v.0.9.8.zip")

```
RECEIVE(sp,Heathinsurance(name,nhsnumber))  
RECEIVE(sp,Nhsrecord(Treatmentphotos(photo,cardiology),Profile(address,Bill(nhsnumber,date))))
```

With the same verification result as above.

Example 8 (Collection Consent)

(“Examples-TEXTMODES/ Examples-TEXTMODE-Cconsent/arch1constextmode.txt” in “Pol and arch files used in the manual v.0.9.8.zip”)

We check whether consent is collected on a piece of data of type *name* (before collecting name).

DATA COLLECTION POLICY

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

Y

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

create:profile

SAVE COLLECTION POLICY & CLOSE

PROVIDER OF DATA

Choose an entity: sp (service provider)

Choose a data group: name

Choose a data type:

SUB-POLICIES: Data Collection

TEXT EDITOR FOR ARCHITECTURE SPECIFICATION (PLEASE PROVIDE ONE ARCH. ACTION PER ROW)

```
RECEIVEAT(sp,Consent(name),Time(t))
RECEIVEAT(sp,Profile(Cv(Personalinfo(name,address),job)),Time(t))
```

VERIFICATION RESULT

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

PRIVACY VIOLATION OF THE POLICY: sp CAN HAVE THE DATA (OF TYPE) : name

The architecture DPR CONFORMS with the policy: sp collects consent before collecting the data (of type) : name

(Before running the conformance verification the user must click on SAVE CONTENT.)

Example 9 (Transfer Consent)

("Examples-TEXTMODES/ Examples-TEXTMODE-FwConsent/arch1fwconstextmode.txt" in "Pol and arch files used in the manual v.0.9.8.zip")

We check whether consent is collected on a piece of data of type *name* by a service provider *sp* (before transferring *name* to an insurance company, *insurance*).

Choose an entity

insurance (company)

sp (service provider)
insurance (company)

Choose a data group

name

Choose a data type

DATA TRANSFER POLICY

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

Y

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

insurance (company)

insurance

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

SAVE TRANSFER POLICY & CLOSE

Choose an entity sp (service provider) Choose a data group name Choose a data type SUB-POLICIES : Data Collection

In the text mode, we define the architecture as

```
RECEIVEAT(sp,Fwconsent(name,insurance),Time(t))
RECEIVEAT(insurance,Profile(Personalinfo(name,address,covertime)),Time(t))
```

TEXT EDITOR FOR ARCHITECTURE SPECIFICATION (PLEASE PROVIDE ONE ARCH. ACTION PER ROW)

```
RECEIVEAT (sp,Fwconsent (name,insurance) ,Time (t) )
RECEIVEAT (insurance,Profile (Personalinfo (name,address,covertime) ) ,Time (t) )
```

VERIFICATION RESULT

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

PRIVACY VIOLATION OF THE POLICY: sp CAN HAVE THE DATA (OF TYPE) : name
PRIVACY VIOLATION OF THE POLICY: insurance CAN HAVE THE DATA (OF TYPE) : name
The architecture DPR CONFORMS with the policy: sp collects consent before forwarding the data (of type) : name

SAVE CONTENT

(Before running the conformance verification the user must click on SAVE CONTENT.)

Example 10 (action CALCULATEFROM/ CALCULATEFROMAT)

(See “Examples-TEXTMODES/ Examples-TEXTMODE-CalculateFrom/” in “Pol and arch files used in the manual v.0.9.8.zip”)

The actions [CALCULATEFROM](#), [CALCULATEFROMAT](#) are supported from the version v.0.9.8 of the tool.

They can be used to specify the action where a component can calculate a piece of data of *Datatype1* from a piece of data of *Datatype2*.

In this example, in the policy we define 3 data types, ephemeral IDs (ephIDa and ephIDb), and the type of random seed, from which the ephemeral IDs are calculated.

The screenshot shows a configuration interface with three dropdown menus: "Choose an entity" (selected: sp (service provider)), "Choose a data group" (selected: seed), and "Choose a data type" (selected: ephIDa, ephIDb, seed). To the right, a "SUB-POLICIES" section contains a row of buttons: Data Collection, Data Usage, Data Storage, Data Retention, Data Transfer, Data Possession, Data Connection (Permit), and Data Connection (Forbid).

Then, we set that *sp* does not have the right to link *ephIDa* and *ephIDb*, and does not have the right to have *ephIDa* and *ephIDb*, and *seed*. We also did not define any consent policy.

The screenshot shows a dialog box titled "DATA CONNECTION/LINKING POLICY - FORBID POLICY". The main text reads: "Choose a (data) group that sp (service provider) is FORBIDDEN to be able to link with the data of type ephIDa". Below this, there is a dropdown menu with "seed" selected. The question "Do you FORBID sp (service provider) only to uniquely link these two types of data ?" is followed by a "No" button. At the bottom, there are three buttons: "ADD DATA CONNECTION POLICY", "DELETE CONNECTION POLICY", and "CLOSE & SAVE". A status bar at the bottom left shows "ephida-ephidb:Any Link is Forbidden".

For the architecture, we define two actions that specify that sp can calculate ephIDa and ephIDb from the random seed. These actions mean that the random seed belongs to the same individual.

TEXT EDITOR FOR ARCHITECTURE SPECIFICATION (PLEASE PROVIDE ONE ARCH. ACTION PER ROW)

```
CALCULATEFROMAT (panel, ephIDa, seed, Time (t))
CALCULATEFROMAT (server, ephIDb, seed, Time (t))
```

SAVE CONTENT

(AFTER OPENNING AN ARCHITECTURE: Before running the conformance verification the user must click on SAVE CONTENT.)

As the verification result, we get:

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 <<MAY NOT>> DPR CONFORM WITH THE POLICY: THE FOLLOWING COLLECTION PURPOSE MAY BE IMPLEMENTED IN THE ARCH, BUT COLLECTION PURPOSE IS NOT DEFINED IN THE POLICY:
==> FOR SOME ENTITY TO << calculatefrom >> A PIECE OF DATA OF TYPE << ephidb >>
- THE ARCHITECTURE <<MAY NOT>> DPR CONFORM WITH THE POLICY: THE FOLLOWING COLLECTION PURPOSE MAY BE IMPLEMENTED IN THE ARCH, BUT COLLECTION PURPOSE IS NOT DEFINED IN THE POLICY:
==> FOR SOME ENTITY TO << calculatefrom >> A PIECE OF DATA OF TYPE << ephida >>
- THE ARCHITECTURE <<MAY NOT>> DPR CONFORM WITH THE POLICY: THE FOLLOWING USAGE PURPOSE MAY BE IMPLEMENTED IN THE ARCH, BUT USAGE PURPOSE IS NOT DEFINED IN THE POLICY:
==> FOR SOME ENTITY TO << calculatefrom >> A PIECE OF DATA OF TYPE << ephidb >>
- THE ARCHITECTURE <<MAY NOT>> DPR CONFORM WITH THE POLICY: THE FOLLOWING USAGE PURPOSE MAY BE IMPLEMENTED IN THE ARCH, BUT USAGE PURPOSE IS NOT DEFINED IN THE POLICY:
==> FOR SOME ENTITY TO << calculatefrom >> A PIECE OF DATA OF TYPE << ephida >>
- PRIVACY VIOLATION OF THE POLICY: sp CAN HAVE THE DATA (OF TYPE) : seed
- PRIVACY VIOLATION OF THE POLICY: sp CAN HAVE THE DATA (OF TYPE) : ephida
- PRIVACY VIOLATION OF THE POLICY: sp CAN HAVE THE DATA (OF TYPE) : ephidb
- PRIVACY VIOLATION OF THE POLICY: sp IS ABLE TO LINK TWO PIECES OF DATA (OF TYPES) : ephida - and - ephidb

The Case of Attackers

Until now, we presented examples of verification where we assumed a benign environment without hostile attackers (and we run the option “VERIFY THE CONFORMANCE ... (NO ATTACKER)”).

In the following, we provide some examples for the verification against the different types of attackers. We start with the external attackers first, then discuss the case of insider attackers and finally, the hybrid attackers.

In these examples, we also demonstrate how DataProVe managed to handle several nested layers of encryptions inside a data type.

We will demonstrate the examples using the TEXTMODE option, but all of these can be done under the GUI mode as well.

COMMON POLICY FOR ALL THE THREE CASES:

(See “Examples-TEXTMODE\Example-TEXTMODE-Attackers\Example-TEXTMODE-External Attackers” in “Pol and arch files used in the manual v.0.9.8.zip”)

In the policy, we define three data types, specifically: 1) name, 2) address, and 3) job (which we defined as UNIQUE, just for checking the unique linkability of *name* and *address*).

The screenshot shows the DataProVe GUI interface. At the top, there is a section for defining a data group. It includes a text input field labeled "PROVIDE A GROUP OF DATA TYPES:" with the value "job". Next to it is a checkbox labeled "IS THIS UNIQUE?" which is checked. To the right is a large empty box labeled "THE DATA TYPES IN THIS GROUP:". A button labeled "ADD DATA GROUP & TYPES" is on the far right. Below this section is a horizontal line. At the bottom, there is a section for selecting an entity and defining sub-policies. It includes three dropdown menus: "Choose an entity" with "sp (service provider)" selected, "Choose a data group" with "address" selected, and "Choose a data type" with an empty dropdown. To the right of these is a section labeled "SUB-POLICIES:" followed by a row of buttons: "Data Collection", "Data Usage", "Data Storage", "Data Retention", "Data Transfer", "Data Possession", "Data Connection (Permit)", and "Data Control (Forbidden)".

We select the default/pre-defined entity **att (attacker)** to specify its data possession and linkability sub-policies. Obviously, we do not allow the attacker to have *name* and *address*, and to link them “uniquely”.

DATA POSSESSION POLICY

Is att (attacker) allowed to have/possess the data group name ? (Y if Yes/Leave empty or N if NO)

SAVE DATA POSSESSION POLICY & CLOSE

Choose an entity att (attacker) Choose a data group name Choose a data type SUB-POLICIES

The same data possession sub-policy is defined for *address* and *job*.

And for linkability:

DATA CONNECTION/LINKING POLICY - FORBID POLICY

Choose a (data) group that att (attacker) is FORBIDDEN to be able to link with the data of type name

address

Do you FORBID att (attacker) only to uniquely link these two types of data ?

Yes

name-address:Unique Link is Forbidden

ADD DATA CONNECTION POLICY
DELETE CONNECTION POLICY
CLOSE & SAVE

Choose an entity att (attacker) Choose a data group name Choose a data type SUB-POLICIES : Dat Collec

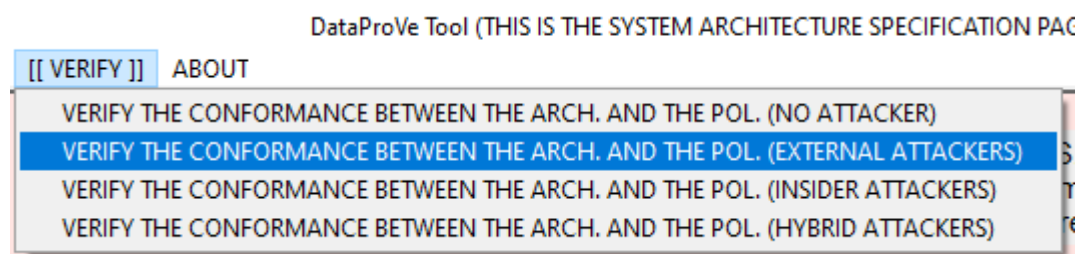
External Attackers

(See “Examples-TEXTMODE\Example-TEXTMODE-Attackers\Example-TEXTMODE-External Attackers” in “Pol and arch files used in the manual v.0.9.8.zip”)

We set the architecture as follows. The service provider can receive an encrypted account that contains an encrypted name and job, with different keys. The service provider can also receive an encrypted skey (2nd action), and a server can receive a nested encryption of a profile that contains a job and address information (note that this message may look weird, but the goal is to demonstrate the nested crypto layers). Finally, we also specify that the attacker can own the keys *key*, *key1*, *key2*, *key3*. We note that the RECEIVE action assumes that the data transmission happens via a public channel available to the attackers for intercepting/eavesdropping.

```
TEXT EDITOR FOR ARCHITECTURE SPECS. (PROVIDE ONE ACTION PER ROW, NO  
Before a conformance verification, click on SAVE CONTENTS  
The same needs to be done before saving an architecture (to save the most  
RECEIVE (sp, Senc (Account (Senc (name, key1) , Senc (job, key2) ) , skey) )  
RECEIVE (sp, Senc (Senc (skey, key3) , key) )  
OWN (att, key)  
OWN (att, key1)  
OWN (att, key2)  
OWN (att, key3)  
RECEIVE (server, Senc (Senc (Senc (Profile (job, address) , key3) , key2) , key1) )
```

Afterwards, we select the second option from the VERIFY tab.



As results, we got that external attacker who can eavesdrop on the data transmission (messages in the RECEIVE action), can have all the three data types, and can link *name* with *address*.

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

NOTE: THE EXTERNAL ATTACKERS ARE NOT PART OF THE SYSTEM. THEY CAN EAVESDROP AND ANALYSE THE COMMUNICATIONS BETWEEN ENTITIES.

- PRIVACY VIOLATION OF THE POLICY: The external attacker CAN HAVE THE DATA (OF TYPE) : name
- PRIVACY VIOLATION OF THE POLICY: The external attacker CAN HAVE THE DATA (OF TYPE) : job
- PRIVACY VIOLATION OF THE POLICY: The external attacker CAN HAVE THE DATA (OF TYPE) : address
- PRIVACY VIOLATION OF THE POLICY: The external attacker CAN !!! UNIQUELY !!! LINK TWO PIECES OF DATA (OF TYPES) : name - and - address

Insider Attackers

(See “Examples-TEXTMODE\Example-TEXTMODE-Attackers\Example-TEXTMODE-Insider Attackers” in “Pol and arch files used in the manual v.0.9.8.zip”)

The policy for this case is the same as the external attacker case above.

For the architecture, now for the four actions OWN, where instead of the attacker, now the server, phone, token, meter entities can own the keys (the rest three actions are unchanged compared to the previous case).

TEXT EDITOR FOR ARCHITECTURE SPECS. (PROVIDE ONE ACTION PER ROW,
Before a conformance verification, click on SAVE CONTI
The same needs to be done before saving an architecture (to save the mo

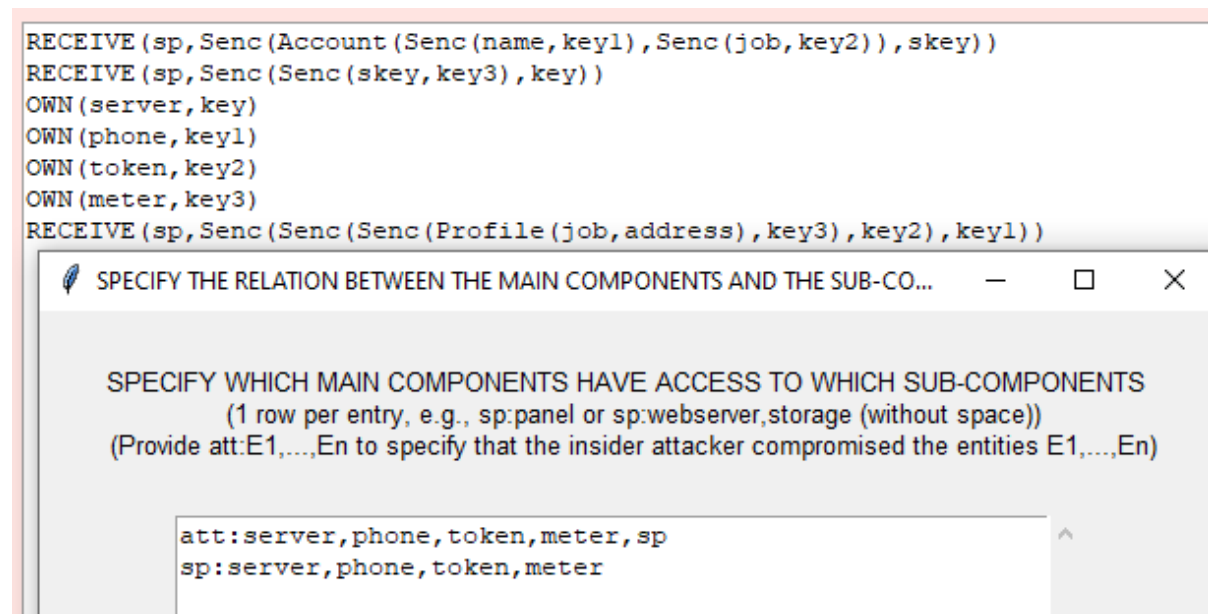
```
RECEIVE (sp, Senc (Account (Senc (name, key1) , Senc (job, key2) ) , skey) )
RECEIVE (sp, Senc (Senc (skey, key3) , key) )
OWN (server, key)
OWN (phone, key1)
OWN (token, key2)
OWN (meter, key3)
RECEIVE (sp, Senc (Senc (Senc (Profile (job, address) , key3) , key2) , key1) )
```

To specify that the attacker compromised the entities *server*, *phone*, *token*, *meter* and *sp*. We use the tab “SPECIFY THE RELATIONSHIP BETWEEN ... (TEXT)”

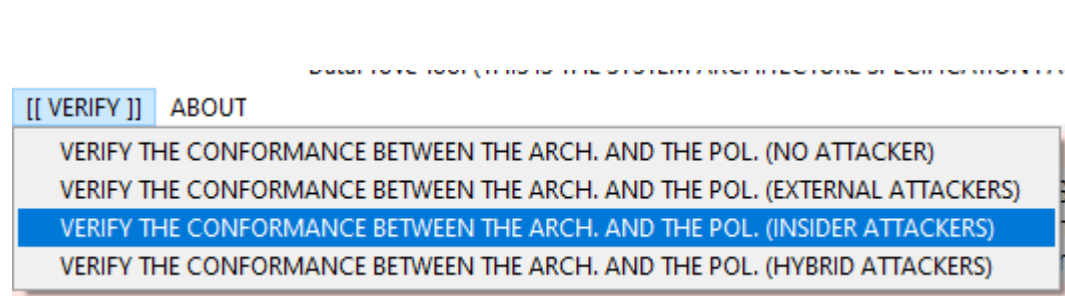
ARCHITECTURE-GUI ARCHITECTURE-TEXTMODE [[VERIFY]] ABOUT

?? HOWTO/HELP/README (TEXT MODE) ??
LAUNCH TEXT EDITOR TO SPECIFY AN ARCHITECTURE
SAVE THE ARCHITECTURE (TEXT)
OPEN AN ARCHITECTURE (TEXT)
SPECIFY THE RELATIONSHIP BETWEEN THE MAIN AND SUB-COMPONENTS (TEXT)

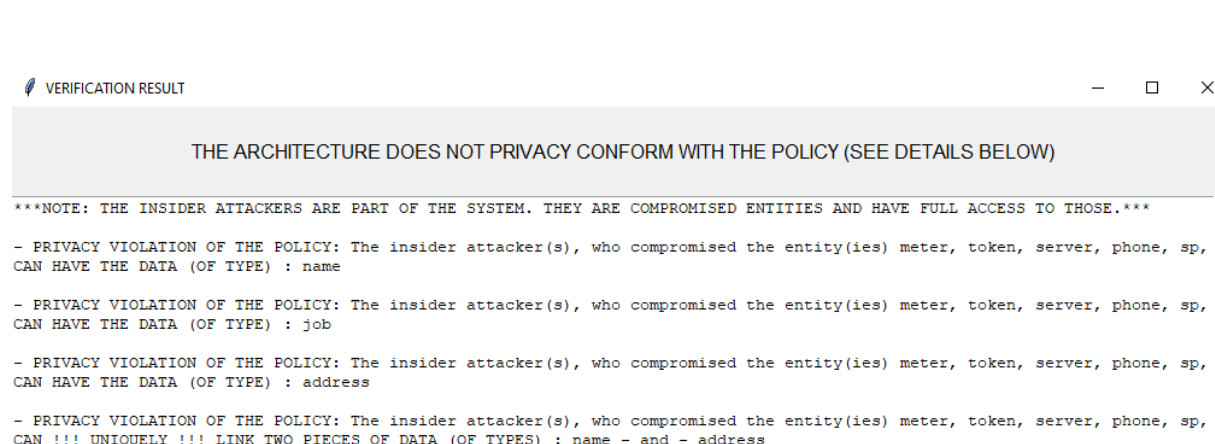
And add the line **att:server,phone,token,meter,sp**.



Afterwards, to run the verification, we select the third option.



As results, we got that external attacker who can eavesdrop on the data transmission (messages in the RECEIVE action), can have all the three data types, and can link *name* with *address*.



Hybrid Attackers

(See “Examples-TEXTMODE\Example-TEXTMODE-Attackers\Example-TEXTMODE-Hybrid Attackers” in “Pol and arch files used in the manual v.0.9.8.zip”)

In this case, we are considering the collusion between external attackers and insider attackers. We also use asymmetric encryption besides the symmetric option. Compared to the previous case, the changes are `OWN(phone,sk(key1))` and `RECEIVE(sp,Aenc(...))`, where `OWN(phone,sk(key1))` says that phone can own a secret key corresponding to the public key, `key1` (this is modelled by the data type `sk(key1)`).

TEXT EDITOR FOR ARCHITECTURE SPECS. (PROVIDE ONE ACTION PER R
Before a conformance verification, click on SAVE C
The same needs to be done before saving an architecture (to save th

```
RECEIVE (sp, Senc (Account (Senc (name, key1) , Senc (job, key2) ) , skey) )  
RECEIVE (sp, Senc (Senc (skey, key3) , key) )  
OWN (server, key)  
OWN (phone, sk (key1) )  
OWN (token, key2)  
OWN (meter, key3)  
RECEIVE (sp, Aenc (Senc (Senc (Profile (job, address) , key3) , key2) , key1) )
```

To specify that the attacker compromised the entities *server*, *phone*, *token*, *meter* and *sp*, again, we use the tab “SPECIFY THE RELATIONSHIP BETWEEN ... (TEXT)”, but now, we change the line **att:server,phone,token,meter,sp** to **att:server,phone,token,meter**, by removing *sp*.

```
RECEIVE (sp, Senc (Account (Senc (name, key1) , Senc (job, key2) ) , skey) )  
RECEIVE (sp, Senc (Senc (skey, key3) , key) )  
OWN (server, key)  
OWN (phone, sk (key1) )  
OWN (token, key2)  
OWN (meter, key3)  
RECEIVE (sp, Aenc (Senc (Senc (Profile (job, address) , key3) , key2) , key1) )
```

SPECIFY THE RELATION BETWEEN THE MAIN COMPONENTS AND THE SUB-CO...

SPECIFY WHICH MAIN COMPONENTS HAVE ACCESS TO WHICH SUB-COMPONENTS
(1 row per entry, e.g., sp:panel or sp:webserver.storage (without space))
(Provide att:E1,...,En to specify that the insider attacker compromised the entities E1,...,En)

att:server,phone,token,meter
sp:server,phone,token,meter

[[VERIFY]] ABOUT

VERIFY THE CONFORMANCE BETWEEN THE ARCH. AND THE POL. (NO ATTACKER)
VERIFY THE CONFORMANCE BETWEEN THE ARCH. AND THE POL. (EXTERNAL ATTACKERS)
VERIFY THE CONFORMANCE BETWEEN THE ARCH. AND THE POL. (INSIDER ATTACKERS)
VERIFY THE CONFORMANCE BETWEEN THE ARCH. AND THE POL. (HYBRID ATTACKERS)

As results, we got the same violations as the previous two cases, as the external attacker can intercept the messages from the RECEIVE action, while the insider attackers compromise the entities *meter*, *phone*, *token*, and *server*, therefore, has access to their data.

VERIFICATION RESULT

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

NOTE: THE HYBRID ATTACKERS CASE SPECIFIES THE COLLUSION BETWEEN INSIDER AND EXTERNAL ATTACKERS.

- PRIVACY VIOLATION OF THE POLICY: The external attacker(s) AND the compromised entity(ies) meter, phone, token, server, CAN HAVE THE DATA (OF TYPE) : name
- PRIVACY VIOLATION OF THE POLICY: The external attacker(s) AND the compromised entity(ies) meter, phone, token, server, CAN HAVE THE DATA (OF TYPE) : job
- PRIVACY VIOLATION OF THE POLICY: The external attacker(s) AND the compromised entity(ies) meter, phone, token, server, CAN HAVE THE DATA (OF TYPE) : address
- PRIVACY VIOLATION OF THE POLICY: The external attacker(s) AND the compromised entity(ies) meter, phone, token, server, TOGETHER CAN !!! UNIQUELY !!! LINK TWO PIECES OF DATA (OF TYPES) : name - and - address