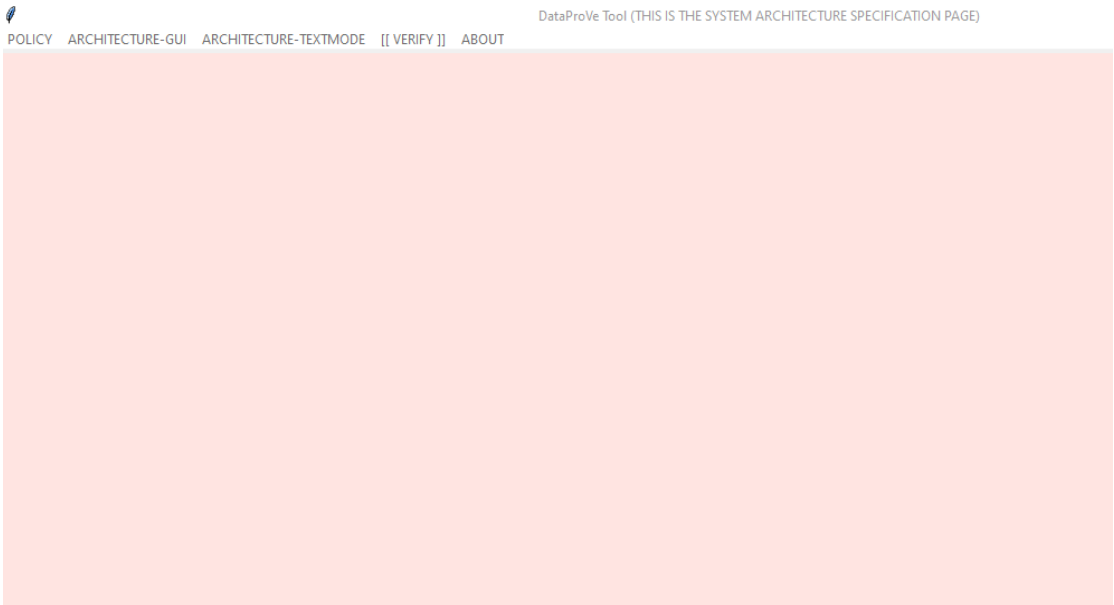
Walkthrough example

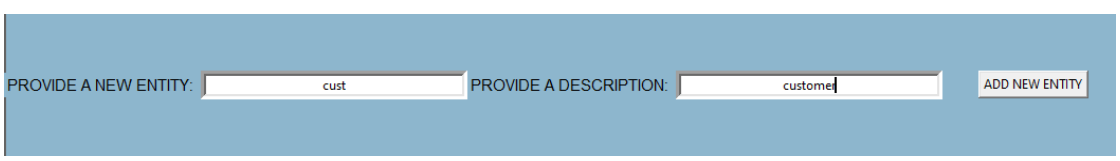
This section provides a simple example to help the user understand how to use the tool step by step. The example is based on a simple smart metering system, where the service provider (sp) collects gas and electric readings, and calculate the bill based on those.

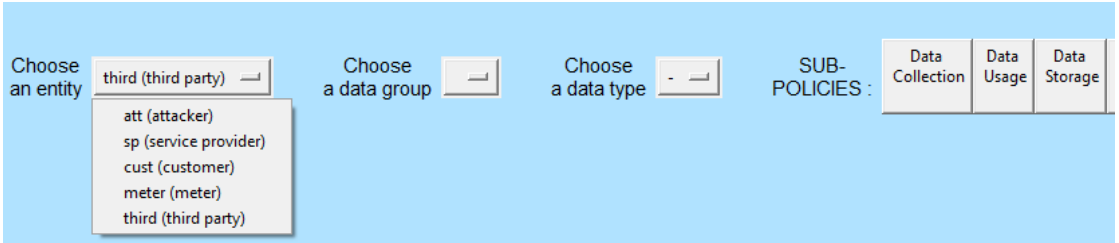
Step 1: After launching the tool, the user will see the architecture specification page (a red colour frame).



Step 2: Choosing the option "Specify a New Data Protection Policy", the users will see a blue frame where they can specify their policy from scratch.

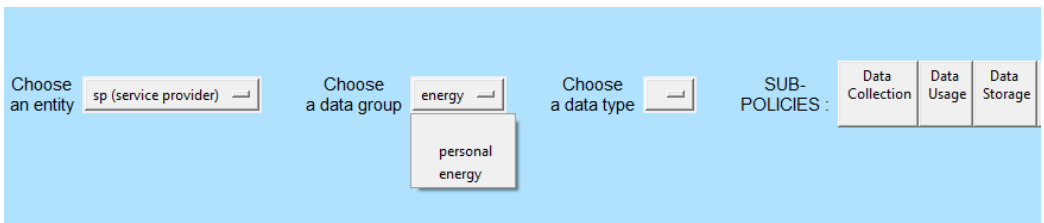
Step 3 (Entities): By default, there are two (built-in) entities, the service provider (sp), and the attacker (att). Let's add two more entities for the smart metering service: the meter (meter) the customer (cust), and the third party (third).





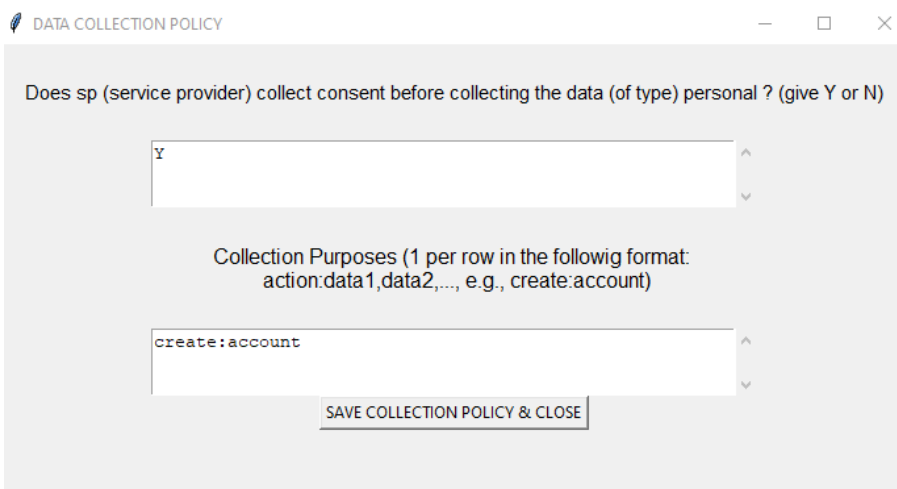
Step 4 (Data types): The next step is to define the data types supported by the service. For simplicity, we define two data types:

1. Personal information (personal) that represents name, address, email, and phone number.
2. Energy reading (energy) that covers gas, electric.



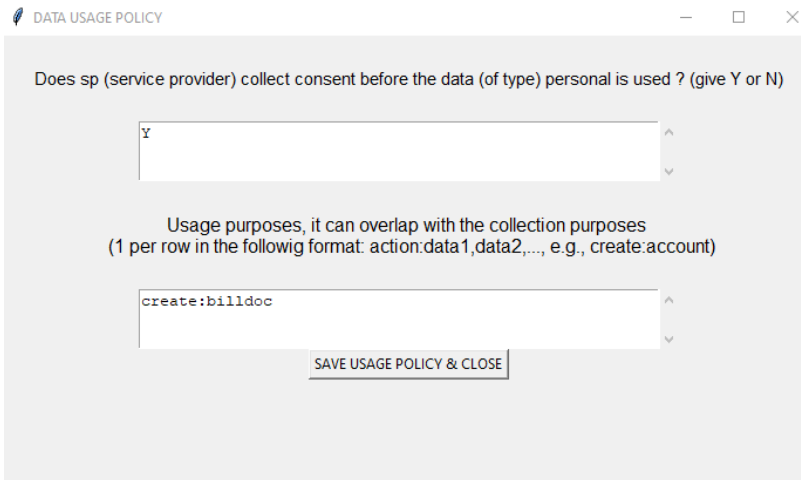
Step 5 (Sub-policies): Now, we can start our sub-policy specifications for each data type.

1. For the data type personal (choose personal in the tab above, then click on "Data Collection"):
   a. the collection sub-policy is (Y, {create:account}), which means that consent is required for collecting this type of data and the purpose of collection is to create an account.

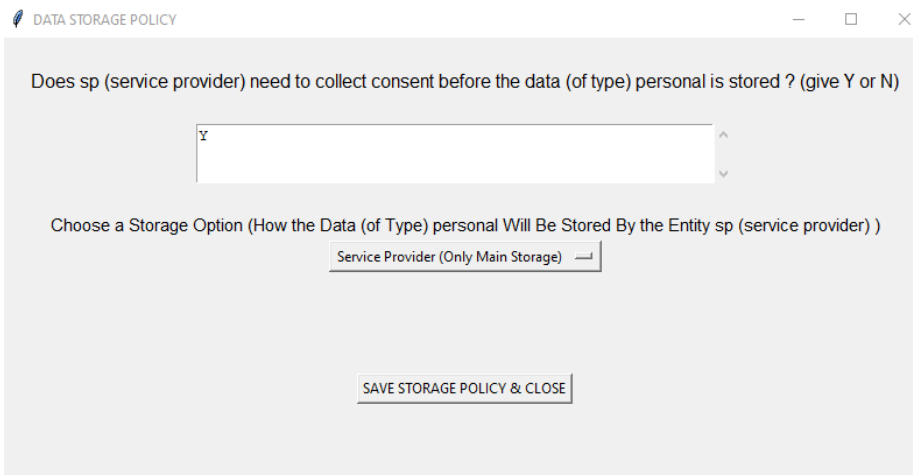


(Click Save & Close to save the sub-policy)

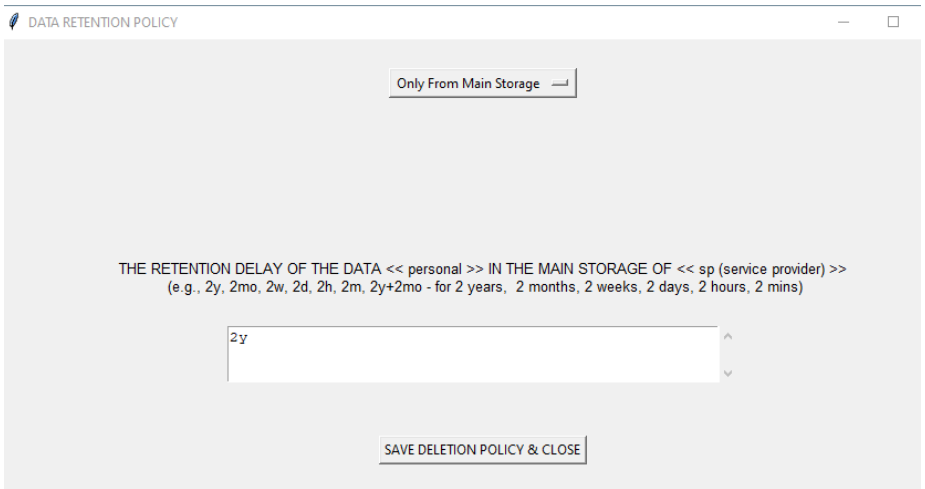
   b. The usage sub-policy is (Y,{create:billdoc}), which means that consent is required to use personal info, with the purpose of creating a bill document.



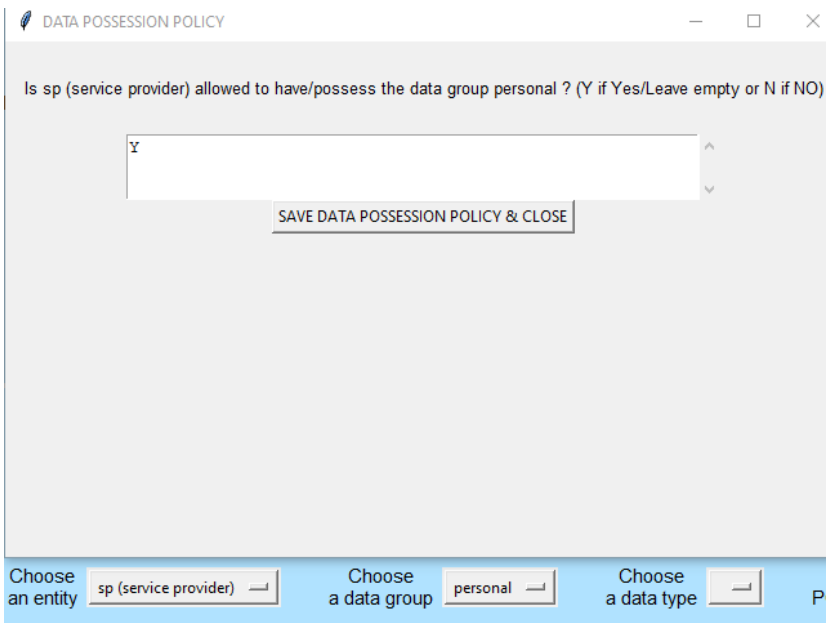
c. The storage sub-policy is (Y,{mainstorage}), which means that the consent is required for data storage. In addition, the data is stored in the main storage place(s) of the service provider.

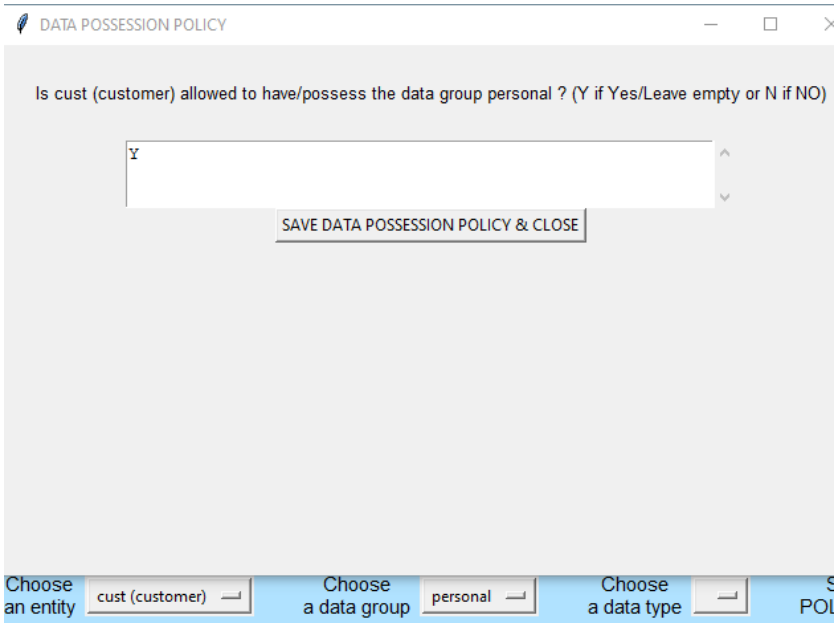


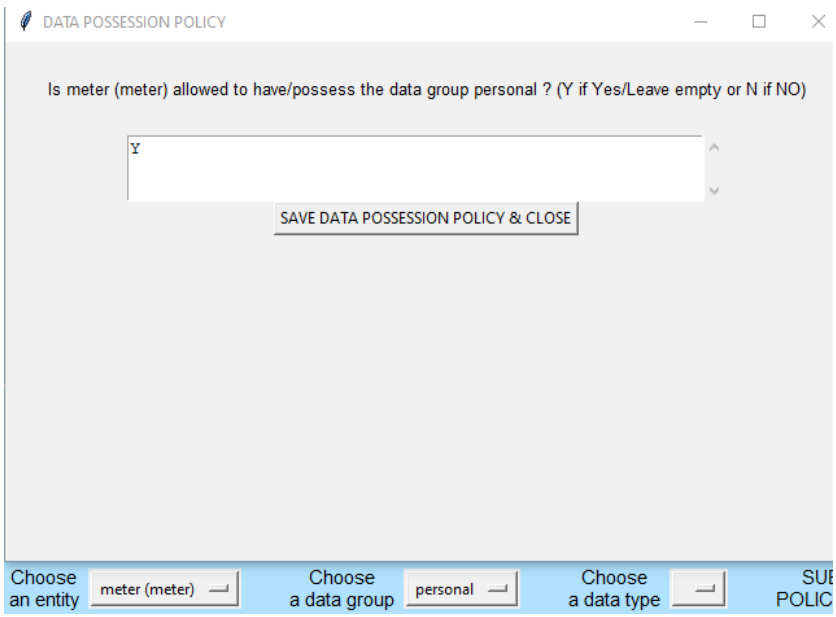
d. The retention sub-policy is ({mainstorage},2y), which means that this type of data is only kept in the main storage at most for 2 years.



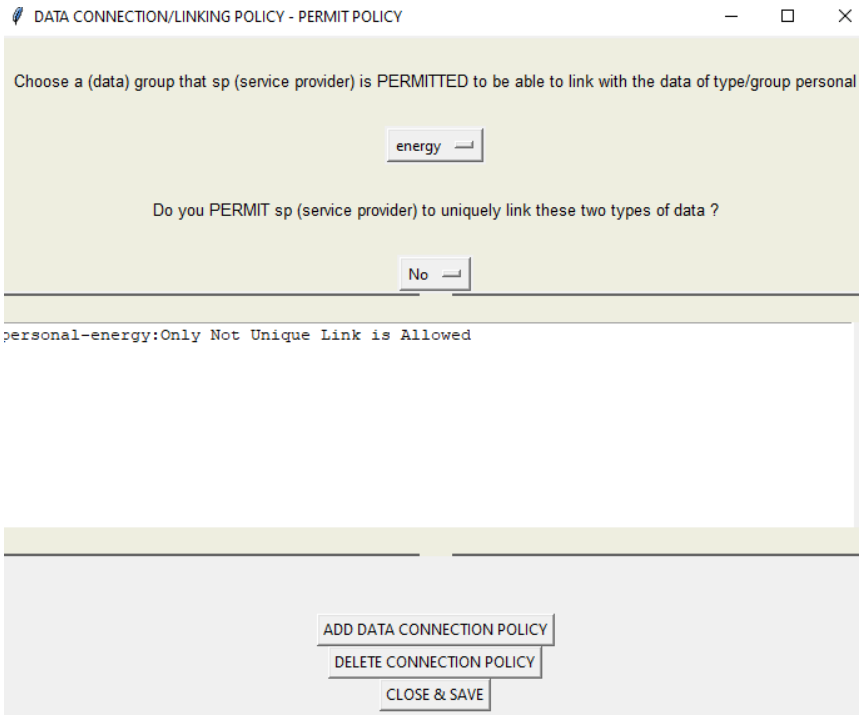
e. For the data transfer policy, we do not allow personal and energy to be forwarded. Hence, the transfer sub-policy is empty.
f. The has sub-policy is {sp, meter, cust}, which means that only sp, meter and cust have the right to access this type of data. This also means that third and att do not have the right to access this type of data.

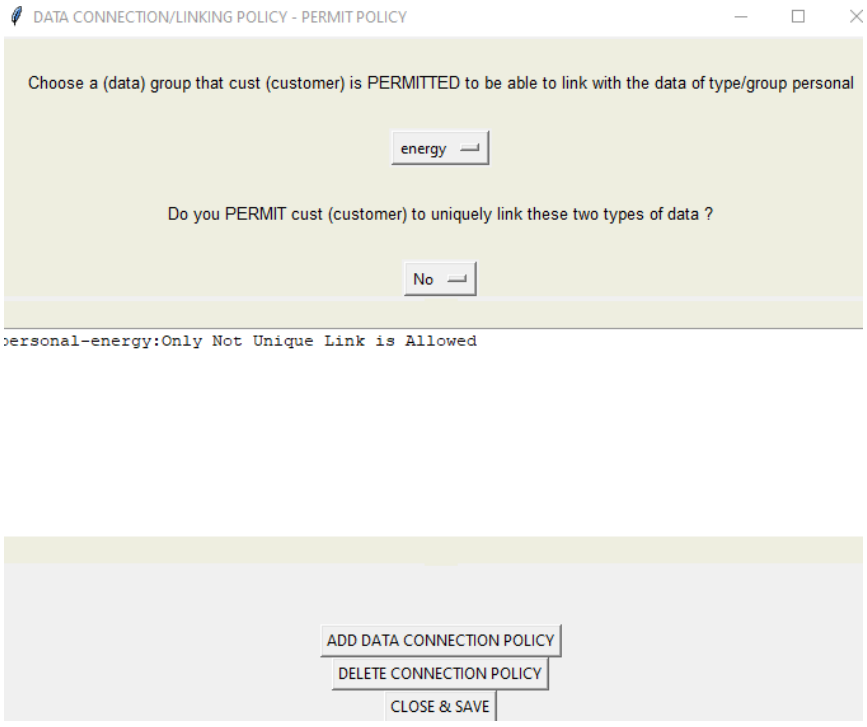




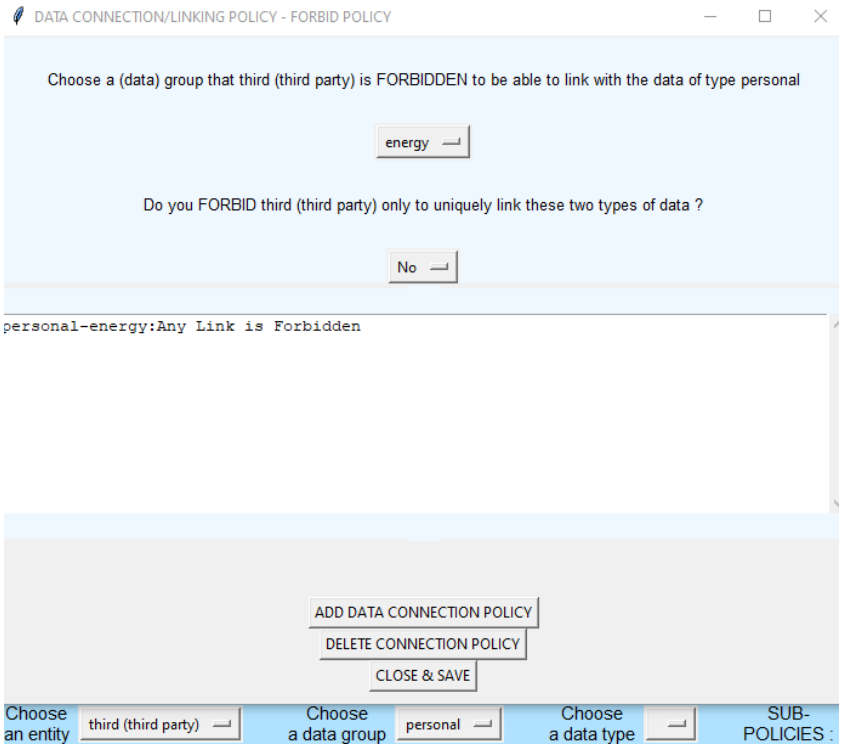


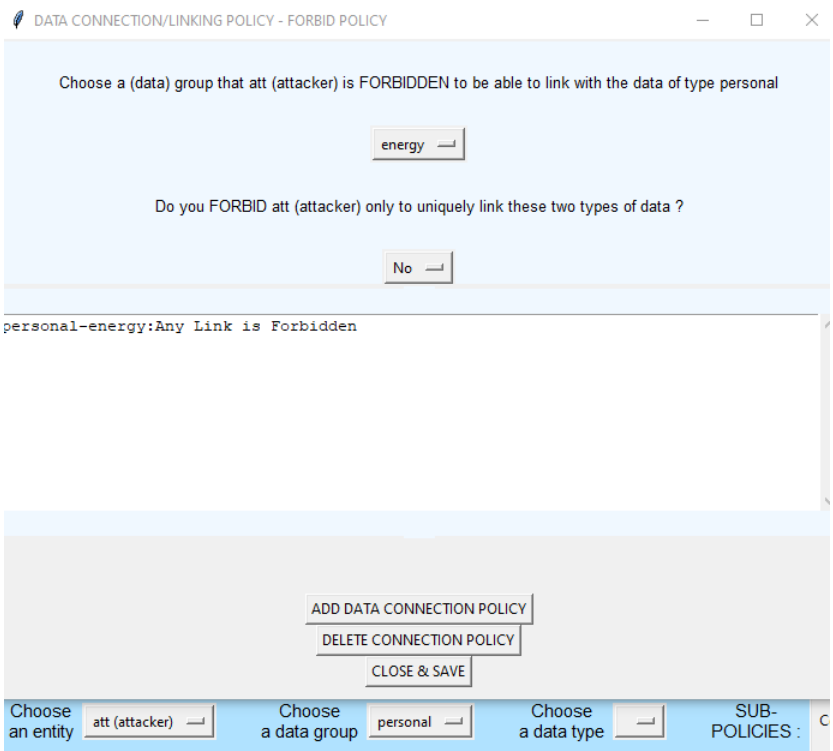
g. The link sub-policy (permit) is {(sp, energy), (cust, energy)}, which means that only sp and cust have the right to link personal with energy (i.e., they know the energy consumption for a given person/address).





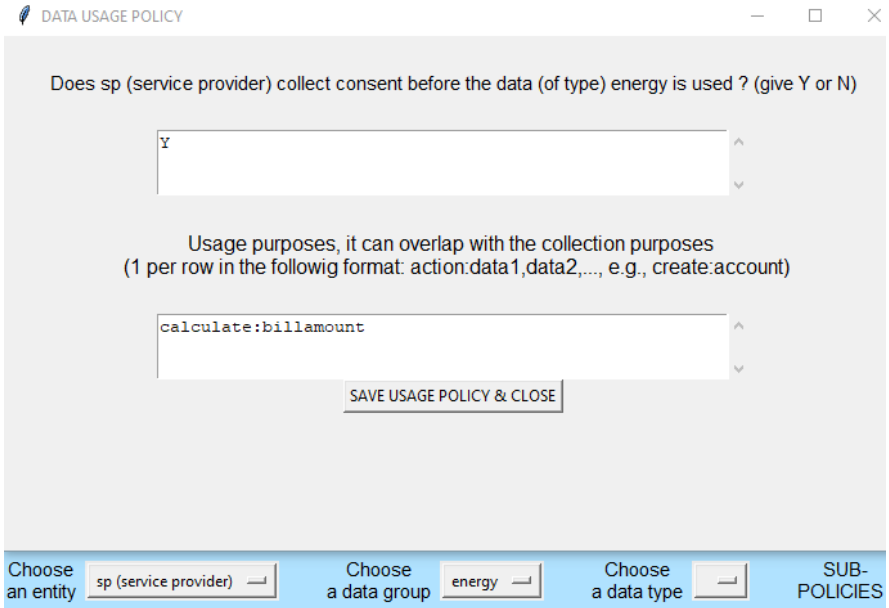
h. The link sub-policy (forbid) is {(third, energy), (att, energy)}, which means that the third party and the attackers do not have the right to link personal with energy.





2. For the data type energy, we define the similar sub-policies like the previous data type, with one exception. Specifically, in the data collection and usage sub-policies,

energy is collected and used for the purpose of calculating the bill amount ({calculate:billamount}).



Save the policy: Under "POLICY" => "SAVE the Policy", the user can save the policy for running a verification, reviewing, and modifying it later.

Step 6 (Architecture specification): Once we finished with the policy specification, we can design and specify the architecture.

We can specify our architecture for the smart metering service either in GUI or Text mode. Let's start with the Text mode.

TEXTUAL MODE: Under the ARCHITECTURE-TEXTMODE tab, we launch the text editor, and start adding the actions line by line.

- RECEIVEAT(sp,energy,Time(t1))
- RECEIVEAT(sp,personal,Time(t2))
- RECEIVEAT(sp,CConsent(energy,sp),Time(t1))
- RECEIVEAT(sp,CConsent(personal,sp),Time(t2))
- RECEIVEAT(sp,UConsent(energy,sp),Time(t1))
- RECEIVEAT(sp,UConsent(personal,sp),Time(t2))
- RECEIVEAT(sp,SConsent(energy,mainstorage),Time(t3))
- RECEIVEAT(sp,SConsent(personal,mainstorage),Time(t4))
- STOREAT(mainstorage,energy,Time(t3))
- STOREAT(mainstorage,personal,Time(t4))
- DELETEWITHIN(mainstorage,energy,Time(2y))
- DELETEWITHIN(mainstorage,personal,Time(2y))
- CREATEAT(sp,Account(personal),Time(t2))
- CREATEAT(sp,BillDoc(personal,energy,BillAmount(energy)),Time(t1))
- CALCULATEAT(sp,BillAmount(energy),Time(t1))
- CALCULATEAT(meter,energy,Time(t8))
- OWN(cust,personal)
- RECEIVEAT(cust,BillDoc(personal,energy,BillAmount(energy)),Time(t9))

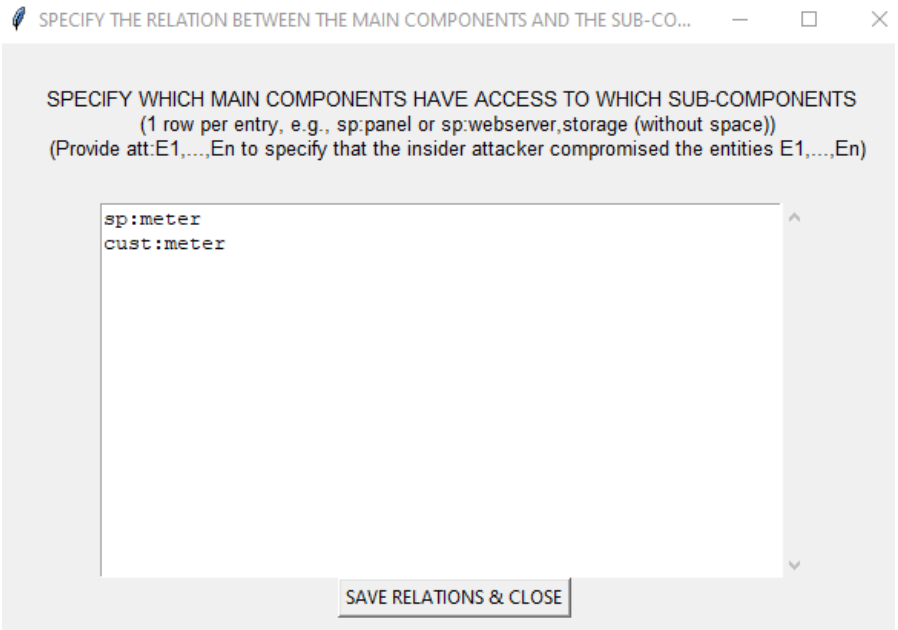
TEXT EDITOR FOR ARCHITECTURE SPECS. (PROVIDE ONE ACTION PER ROW, NO PUNCTUATION AT THE END)
Before a conformance verification, click on SAVE CONTENT.
The same needs to be done before saving an architecture (to save the most up-to-date version).

```
RECEIVEAT(sp,energy,Time(t1))
RECEIVEAT(sp,personal,Time(t2))
RECEIVEAT(sp,CConsent(energy,sp),Time(t1))
RECEIVEAT(sp,CConsent(personal,sp),Time(t2))
RECEIVEAT(sp,UConsent(energy,sp),Time(t1))
RECEIVEAT(sp,UConsent(personal,sp),Time(t2))
RECEIVEAT(sp,SConsent(energy,mainstorage),Time(t3))
RECEIVEAT(sp,SConsent(personal,mainstorage),Time(t4))
STOREAT(mainstorage,energy,Time(t3))
STOREAT(mainstorage,personal,Time(t4))
DELETEWITHIN(mainstorage,energy,Time(2y))
DELETEWITHIN(mainstorage,personal,Time(2y))
CREATEAT(sp,Account(personal),Time(t2))
CREATEAT(sp,BillDoc(personal,energy,BillAmount(energy)),Time(t1))
CALCULATEAT(sp,BillAmount(energy),Time(t1))
CALCULATEAT(meter,energy,Time(t8))
OWN(cust,personal)
RECEIVEAT(cust,BillDoc(personal,energy,BillAmount(energy)),Time(t9))
```

SAVE CONTENT

Click “SAVE CONTENT”. The architecture in text mode needs to save before conformance verification.

The next step is to <u>set the relationship between</u> the entities sp and meter, and cust and meter. Under the tab “ARCHITECTURE-TEXTMODE”, we choose “SPECIFY THE RELATIONSHIP BETWEEN THE MAIN AND SUB-COMPONENTS (TEXT)”, which opens a window where we can provide two lines: sp:meter and cust:meter. This means that sp and cust have access to the reading of meter.



Save the architecture: Under "ARCHITECTURE-TEXTMODE" => "SAVE the Architecture (Text)", the user can save the architecture for running a verification, reviewing, and modifying it later.

Step 7 (Conformance verification):

Once we are ready with the policy and architecture specifications, we can run the conformance verification. We can choose among four options.

1. Run the verification without the presence of any attacker (e.g., the entity att is ignored).

```
- The architecture functionally conforms with the policy: sp can have the data (of type) personal
- The architecture functionally conforms with the policy: cust can have the data (of type) personal
- The architecture DOES NOT functionally conform with the policy: meter CANNOT have the data (of type) personal
- The architecture functionally conforms with the policy: sp can have the data (of type) energy
- The architecture functionally conforms with the policy: meter can have the data (of type) energy
- The architecture functionally conforms with the policy: cust can have the data (of type) energy
- The architecture functionally conforms with the policy: sp can link two pieces of data (of type) : personal - and - energy
- The architecture functionally conforms with the policy: cust can link two pieces of data (of type) : personal - and - energy
- The architecture DPR CONFORMS with the policy: consent can be collected by << sp >> before collecting the data (of type) : personal
- The architecture DPR CONFORMS with the policy: consent can be collected by << sp >> before collecting the data (of type) : energy
- The architecture DPR CONFORMS with the policy: consent can be collected by << sp >> before using the data (of type) : personal
- The architecture DPR CONFORMS with the policy: consent can be collected by << sp >> before using the data (of type) : energy
- The architecture DPR CONFORMS with the policy:  consent can be collected by << sp >> before storing the data (of type) : personal
- The architecture DPR CONFORMS with the policy:  consent can be collected by << sp >> before storing the data (of type) : energy
```

2. Run the verification to check if the has and link sub-policies are violated assuming the external attacker(s).

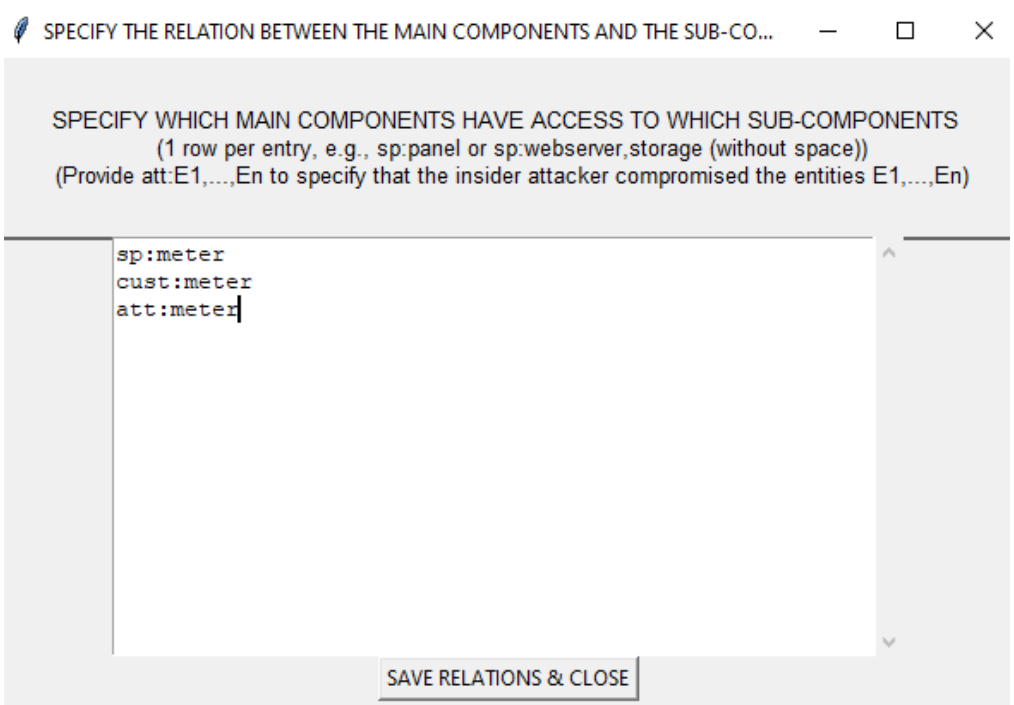
```
***NOTE: THE EXTERNAL ATTACKERS ARE NOT PART OF THE SYSTEM. THEY CAN EAVESDROP AND ANALYSE THE COMMUNICATIONS BETWEEN ENTITIES.***

- PRIVACY VIOLATION OF THE POLICY: The external attacker CAN HAVE THE DATA (OF TYPE) : personal

- PRIVACY VIOLATION OF THE POLICY: The external attacker CAN HAVE THE DATA (OF TYPE) : energy

- PRIVACY VIOLATION OF THE POLICY: The external attacker CAN LINK TWO PIECES OF DATA (OF TYPES) : personal - and - energy
```

   Since the messages exchanged are unencrypted, the external attacker who eavesdrop on the communication can obtain personal, energy and link personal to energy due to the bill document (BillDoc).

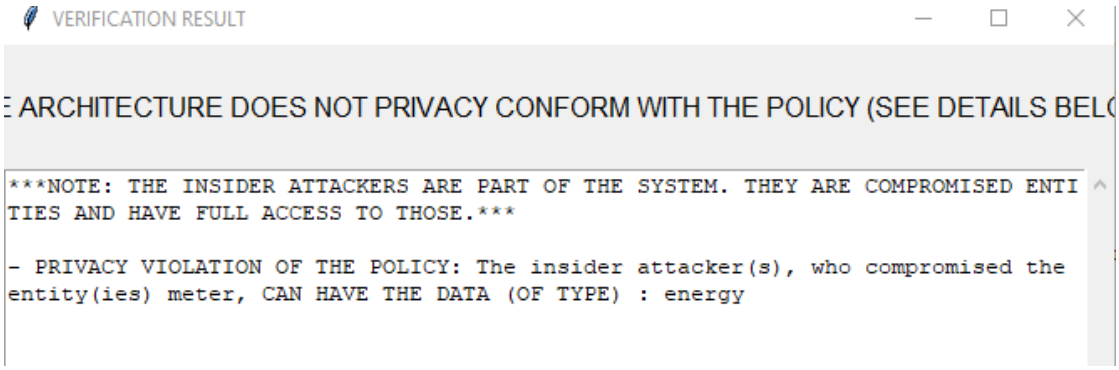
3. Run the verification to check if the has and link sub-policies are violated assuming (only) the insider attacker(s).

   To verify against insider attackers, we need to define which entity/component has been compromised by the attacker (that has full access to the compromised entity). For example, we specify that the attacker has access to meter.

   Under "SPECIFY THE RELATIONSHIP BETWEEN THE MAIN AND SUB-COMPONENTS (TEXT)", we add: att:meter.

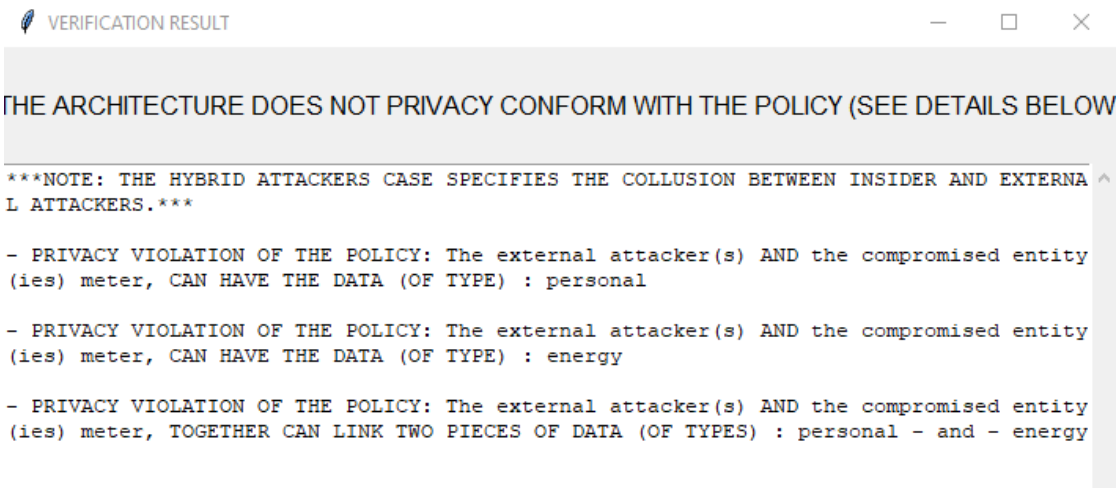


   As the verification result, we get that the insider attacker only have access to energy (because of the compromised meter), but not to personal.



4. Run the verification to check if the has and link sub-policies are violated assuming the hybrid attacker(s).

   The hybrid attacker case is the combination of the external and insider attackers.



Step 8 (optional):
We can make changes to our architecture so that it is secured against external attackers, by applying encryption on the exchanged messages. To do this, we change the actions (directly in the text editor) to, for example:

- `RECEIVEAT(sp,Senc(energy,key),Time(t1))`
- `RECEIVEAT(sp,Senc(personal,key),Time(t2))`
- `RECEIVEAT(cust,Senc(BillDoc(personal,energy,BillAmount(energy)),key),Time(t9))`
- `RECEIVEAT(sp,CConsent(energy,sp),Time(t1))`
- `RECEIVEAT(sp,CConsent(personal,sp),Time(t2))`
- `RECEIVEAT(sp,UConsent(energy,sp),Time(t1))`
- `RECEIVEAT(sp,UConsent(personal,sp),Time(t2))`
- `RECEIVEAT(sp,SConsent(energy,mainstorage),Time(t3))`
- `RECEIVEAT(sp,SConsent(personal,mainstorage),Time(t4))`
- `STOREAT(mainstorage,energy,Time(t3))`

- STOREAT(mainstorage,personal,Time(t4))
- DELETEWITHIN(mainstorage,energy,Time(2y))
- DELETEWITHIN(mainstorage,personal,Time(2y))
- CREATEAT(sp,Account(personal),Time(t2))
- CREATEAT(sp,BillDoc(personal,energy,BillAmount(energy)),Time(t1))
- CALCULATEAT(sp,BillAmount(energy),Time(t1))
- CALCULATEAT(meter,energy,Time(t8))
- OWN(cust,personal)
- OWN(cust,key)
- OWN(sp,key)

After clicking "SAVE CONTENT", we can run the verification against **external attackers** to see the difference in the result.