SUNFISH Blockchain: API References

Internal Use Only

SOTON SUNFISH Team

14 December, 2016

1. **Introduction**

This document outlines the specifications of the APIs used to interface with the Blockchain component (the Registry Interface, RI) within the SUNFISH platform. The RM exposes several endpoints that can be used to perform a specific functionality. The list of available functionalities have been discussed in the SUNFISH blockchain proposal document and briefly discussed in the next section. Each endpoint essentially provides a REST interface which expects any request to contain specific parameters and produces responses after executing the predefined functionality with the supplied parameters. The specification for each API is presented in the following section.

1. **Blockchain-integrated SUNFISH Platform**

A blockchain-integrated SUNFISH platform has been proposed in the SUNFISH blockchain proposal document which was circulated in early October. In that document, it has been envisioned that the following functionalities will exploit the Ethereum blockchain technology.

1. Access control monitoring
2. Storage of the state of the federation
3. Distributed storage of access control policies
4. Support for Data Masking
5. Support for Anonymization
6. Distributed storage of SLA
7. Service Agreement Contract

A general architecture to deploy such functionalities is illustrated in Figure 1.

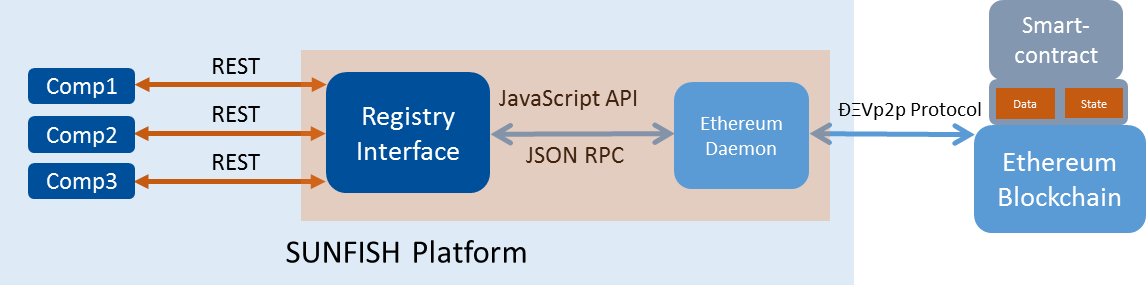


Figure 1: SUNFISH Blockchain architecture

The Registry Interface (RI) is essentially a DApp (Decentralised Application) which in one hand exposes REST-based APIs and on the other hand interacts with the smart-contract(s) in the blockchain. To perform each specified functionality, we envision that there will be a separate smart-contract and corresponding a REST API exposing a REST endpoint. Next, we present the specification for each REST endpoint.

1. **API Specification**
   1. Access control monitoring

Currently, FRM utilises the Ethereum blockchain to store access logs. The API supports two operations, store and read, exposed by two endpoints. The API specification for each endpoint is presented below.

Table 1 – Monitoring API: Store operation

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/monitoring/store | This endpoint is used to store relevant monitoring data. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "loggerID": {  "type": "string"  },  "timeStamp": {  "type": "string"  },  "token": {  "type": "string"  },  "dataType": {  "type": "string"  },  "data": {  "type": "string"  }  },  "required": [  "loggerID",  "timeStamp",  "token",  "dataType",  "data"  ]  } |
| Parameter explanation | loggerID: Identifier of the requesting entity.  timeStamp: The timestamp in the following format - "2006-01-02 15:04:05".  token: A crypto token to validate if the entity is allowed to perform the requested operation.  dataType: {REQUEST|RESPONE}, can be one of these values.  Data: Base64 encoded XACML request/response. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data along with the parameters have been successfully stored in the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains an index in JSON format signifying the index of the stored values. This index is required to retrieve this particular set of values. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "index": {  "type": "integer"  }  },  "required": [  "index"  ]  } |

Table 2 – Monitoring API: Read operation

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/monitoring/read | This endpoint is used to retrieve the stored monitoring data using the index. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "loggerID": {  "type": "string"  },  "token": {  "type": "string"  },  "index": {  "type": "integer"  }  },  "required": [  "loggerID",  "token",  "index"  ]  } |
| Parameter explanation | loggerID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  index: Index of the requested data as provided during the store operation. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data has been successfully read from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a set of data retrieved using the index in JSON format. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "loggerID": {  "type": "string"  },  "timeStamp": {  "type": "string"  },  "token": {  "type": "string"  },  "dataType": {  "type": "string"  },  "data": {  "type": "string"  }  },  "required": [  "loggerID",  "timeStamp",  "token",  "dataType",  "data"  ]  } |
| Explanation | As specified above. |

There is no API specified for the update or delete operation for this functionality. This is to ensure that no one can alter the data once it has been logged.

* 1. Storage of the state of the federation

Currently, the state of each participant to the federation is assumed to be stored in a consistent form by the Registry. We envision that the data planned to be part of the registry (i.e., a list of the available services and their providers) is stored in the blockchain via its corresponding smart-contract. We envision that the API will support three operations - store, read, update and delete - exposed by three endpoints. The API specification for each endpoint is presented below.

Table 3 – State Storage API: Store operation

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/state/store | This endpoint is used to store relevant state information. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "serviceID": {  "type": "string"  },  "url": {  "type": "string"  },  "name": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "serviceID"  "url",  "name"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  serviceID: The identifier of the service.  url: The URL of the service.  name: The name of the service. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data along with the parameters have been successfully stored in the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a message in JSON format signifying that the state information has been stored successfully. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "message": {  "type": "string"  }  },  "required": [  "message"  ]  } |

Table 4 – State storage API: Read operation

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/state/read | This endpoint is used to retrieve the state information using the serviceID. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "serviceID": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "serviceID"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  serviceID: The identifier of the service. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data has been successfully read from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a set of data retrieved using the index in JSON format. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "serviceID": {  "type": "string"  }  "url": {  "type": "string"  },  "name": {  "type": "string"  }  },  "required": [  "serviceID",  "url",  "name"  ]  } |
| Explanation | As specified above. |

Table 5 – State storage API: Update operation

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/state/update | This endpoint is used to update a particular state information using the serviceID. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "serviceID": {  "type": "string"  },  "list": {  "type": "array",  "items": {  "type": "object",  "properties": {  "name": {  "type": "string"  },  "value": {  "type": "string"  }  },  "required": [  "name",  "value"  ]  }  }  },  "required": [  "requestorID",  "token",  "serviceID",  "list"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  serviceID: The identifier of the service.  list: Reprents a list consisting of names and values of attributes  name: The name of the attribute whose value needs to be updated.  value: The new value. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data has been successfully read from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a message in JSON format signifying that the state information has been updated successfully. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "message": {  "type": "string"  }  },  "required": [  "message"  ]  } |
|  |

Table 6 – State Storage API: Delete operation

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/state/delete | This endpoint is used to remove the state information identified by the serviceID. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "serviceID": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "serviceID"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  serviceID: Universally unique identifier for a service. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data along with the parameters have been successfully de-ferenced in the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a message in JSON format signifying that the state information has been deleted successfully. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "message": {  "type": "string"  }  },  "required": [  "message"  ]  } |

It is to be noted that the delete operation will de-reference (nullify) the particular state information, as identified by the serviceID. What it means is that if a read operation is performed to retrieve the state information as identified by the serviceID, it will return a message which will signify that the state information has been removed.

* 1. Distributed storage of access control policies

We envision that access control policies will be stored in a distributed fashion in the blockchain. This means that the blockchain will be utilised as a policy repository. In this regard we propose that only the PRP would require to interact with the blockchain via its respective RI so that the PRP can store, retrieve, update and remove policies. In this way, the APIs defined for the PAP remain unchanged. This functionality exposes several endpoints in order to realise the functionalities of the PRP. The API specification of each endpoint is based on the API specification as described in D4.5. The existing specification has been slightly modified to match our vision of utilising blockchain. The modification itself has been as minimum as possible to ensure that a minimum amendment in the existing deployment is required. The API specification for each endpoint is presented below.

Table 7 – PRP: Storing a policy

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/prp/policy/store | This endpoint is used to store a new a policy. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "policy": {  type": "string"  },  "version": {  type": "string"  },  "expirationTime": {  type": "string"  },  "id": {  type": "string"  },  "serviceID": {  type": "string"  },  "policyType": {  type": "string"  }  },  "required": [  "requestorID",  "token",  "policy",  "version",  "id",  "expirationTime"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  policy: Base64-encoded policy data.  version: The version of the policy.  expirationTime: Specifies the expiration time of the policy in milli-seconds starting from midnight, January 1, 1970 UTC. After this time the policy set must not be used.  id: An identifer for the policy.  serviceID (optional): The identifier of the service this policy applies to.  policyType (optional): The type of the policy. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data has been successfully stored in the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a message in JSON format signifying that the policy has been successfully stored. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "message": {  "type": "string"  }  },  "required": [  "message"  ]  } |
|  |

While the policy is being stored, we suggest to create another record, as part of the state information, which will contain the list of services for which this particular policy has been assigned to. That is why, the API also requires the identifier of the service for storing a policy. There might be some policies (e.g. administrative policies) which might not be tied to a particular service. The absence of “serviceID” in the parameter of the request will be used to indicate this scenario. In such cases, the policy will not be added to the list of services.

Table 8 – PRP: Retrieving a policy by id and version

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/prp/policy/read | This endpoint is used to retrieve a policy by an id and version. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "id": {  "type": "string"  },  "version": {  "type": "string"  }  "rootPolicy": {  type": "boolean"  }  },  "required": [  "requestorID",  "token",  "id",  "version",  "rootPolicy"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  id: Specifies the id of the policy to be returned in the response.  version: Specifies the version of the policy to be returned in the response.  rootPolicy: {true|false}, defines if a root policy or a re-usable policy should be returned. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data has been successfully read from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.).  **404:** The policy set with the specified id was not found. |
| Body | The response body for a successful response (response code 200) contains the requested policy along with the expiration time, all in JSON. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "policy": {  "type": "string"  },  "expirationTime": {  "type": "integer"  }  },  "required": [  "policy",  "expirationTime"  ]  } |
| Explanation | policy: BASE64-enocded data of the requested policy.  expirationTime: Specifies the expiration time of the policy set in milliseconds starting from midnight, January 1, 1970 UTC. After this time the policy set must not be used. |

Table 8 – PRP: Retrieving a set of policies by a service identifier

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/prp/policy/service | This endpoint is used to retrieve a set of policies using a service identifier. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "serviceID": {  "type": "string"  },  "policyType": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "serviceID",  "policyType"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  serviceID: Specifies the service identifier to be used to retrieve the set of policies belonging to that identifier.  policyType: The type of the policy. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data has been successfully read from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.).  **404:** The policy set with the specified id was not found. |
| Body | The response body for a successful response (response code 200) contains the requested set of policcies belonging to that particular service identifier, all in JSON. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "list": {  "type": "array",  "items": {  "type": "object",  "properties": {  "id": {  "type": "string"  },  "version": {  "type": "string"  },  "policy": {  "type": "string"  }  },  "required": [  "id",  "version",  "policy"  ]  }  }  },  "required": [  "list"  ]  } |
| Explanation | list: Reprents a list where each element consists of an id, version and policy.  id: Specifies the id of the policy.  version: Specifies the version of the policy.  policy: BASE64-enocded data of the policy. |

Table 9 – PRP: Updating a policy by id and version

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/prp/policy/update | This endpoint is used to update a policy by an id and version. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "id": {  "type": "string"  },  "policy": {  "type": "string"  },  "version": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "id",  "policy",  "version"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  id: Specifies the id of the policy to be updated in the response.  policy: Base64-encoded policy data.  version: Specifies the version of the policy. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data has been successfully read from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.).  **404:** The policy set with the specified id was not found. |
| Body | The response body for a successful response (response code 200) contains a message in JSON format signifying that the policy has been successfully updated. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "message": {  "type": "string"  }  },  "required": [  "message"  ]  } |
| Explanation | message: A message signifying that the policy has been updated successfully. |

Table 10 – PRP: Removing a policy by id

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/prp/policy/delete | This endpoint is used to remove a policy by an id and version. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "id": {  "type": "string"  },  "version": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "id",  "version"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  id: Specifies the id of the policy to be removed.  version: Specifies the version of the policy to be removed. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data has been successfully read from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.).  **404:** The policy set with the specified id was not found. |
| Body | The response body for a successful response (response code 200) contains a message in JSON format signifying that the policy has been successfully updated. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "message": {  "type": "string"  }  },  "required": [  "message"  ]  } |
| Explanation | message: A message signifying that the policy has been removed successfully. |

We would like to propose a restriction for removing a particular policy. Our idea is that a policy can be removed only if no service is being offered that relies on the corresponding policy. When a request is submitted to remove a policy, the corresponding check will be carried out by retrieving the required list from the state information created during the policy creation phase.

It is to be noted that, a successful delete operation will de-reference (nullify) every information belonging to the particular policy ID.

We have not defined any API for retrieving any policy set. This is because we believe the PRP can realise this functionality by making iterating calls to retrieve all policies, belonging to a particular policy set, using their ids.

* 1. Support for data masking

Data masking component utilises blockchain and cloud to store masking tables, in order to improve the confidentiality, integrity and availability of masking tables. More specifically, data masking is still carried out in the segregated environment where masking tables are encrypted with the public key of an authorised data consumer. Then, the encrypted masking tables are divided into pieces where each piece is replicated and uploaded to different clouds/servers. The blockchain stores the addresses where each piece is stored. Finally, authorised data consumers retrieve the encrypted masking tables and carry out the unmasking locally, using their private keys.

The API supports various operations on (i) the keys of authorised data consumers, (ii) encrypted masking tables, and (iii) pieces that generated from the encrypted masking tables. The API specification for each endpoint is presented below.

1. **Keys**

Table 11 – Keys: Register a public key

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/keys/register | This endpoint is used to register a public key for an authorised data consumer[[1]](#footnote-1) account. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "userID": {  "type": "string"  },  "key": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "userID",  "key"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  userID: Identifier of an authorised data consumer.  key: public key to be registered. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data along with the parameters have been successfully stored in the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a message in JSON format signifying that the key has been registered successfully. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "message": {  "type": "string"  }  },  "required": [  "message"  ]  } |
|  |

Table 12 – Keys: Read a key

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/keys/read | This endpoint is used to retrieve the public key associated with a user. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "userID": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "userID"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  userID: Identifier of a user. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data has been successfully read from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a set of data retrieved using the index in JSON format. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "userID": {  "type": "string"  },  "key": {  "type": "string"  }  },  "required": [  "userID",  "key"  ]  } |
|  |  |

Table 13 – Keys: Delete a key

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/keys/delete | This endpoint is used to destroy a public key given a userID. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "userID": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "userID"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  userID: Identifier of a user. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The key has been successfully deleted from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a message in JSON format signifying that the key has been deleted successfully. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "message": {  "type": "string"  }  },  "required": [  "message"  ]  } |
|  |  |

Again, the delete operation will de-reference (nullify) the particular key information, as identified by the userID. Then if a read operation is performed to retrieve the key information as identified by the userID, it will return a message which will signify that the key information has been removed.

Table 14 – Keys: Update a key

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/keys/update | This endpoint is used to update a public key given user’s ID. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "userID": {  "type": "string"  },  "key": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  “userID”,  "key"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  key: new public key.  userID: identifier of a user. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The key has been successfully updated in the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a message in JSON format signifying that the key has been updated successfully. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "message": {  "type": "string"  }  },  "required": [  "message"  ]  } |
|  |  |

1. **Tables**

We see an encrypted masking table as a bookcase which contains the pieces that are generated from the table.

Table 15 – Tables: Create an empty table

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/tables | This endpoint is used to create a table. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "timestamp": {  "type": "string"  },  "dataID": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "timestamp",  "dataID"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  dataID: Identifier of the dataset that the encrypted masking table relates to.  timestamp: The timestamp in the following format - "2006-01-02 15:04:05". |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The table has been successfully created in the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains an index in JSON format signifying the index of the created table, denoted as tableID. The index is required to retrieve this particular table. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "tableID": {  "type": "string"  }  },  "required": [  "tableID"  ]  } |
|  |  |  |

Table 16 – Tables: Read a table

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/tables /read | This endpoint is used to retrieve a stored table by its unique ID. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "tableID": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "tableID"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  tableID: Identifier of an encrypted masking table. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The table has been successfully read from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a set of data retrieved using the index in JSON format. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "tableID": {  "type": "string"  },  "timestamp": {  "type": "string"  },  "pieces": {  "type": "array",  "items": {  "type": “object”,  "properties": {  "pieceID": {  "type": "string"  },  "position": {  "type": "string"  }  }  }  }  },  "required": [  "tableID",  "timestamp",  "pieces"  ]  } |
| Explanation | pieceID: identifier of a piece.  position: the position where the piece is located in the table. |

Note that “pieceID” and “position” in “pieces” can be empty strings if a specific piece has not been added to the table yet.

Table 17 – Tables: Delete a table

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/tables/delete | This endpoint is used to destroy a table by its unique ID. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "tableID": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "tableID"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  tableID: Identifier of an encrypted masking table. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The table has been successfully deleted from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a message in JSON format signifying that the table has been deleted successfully. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "message": {  "type": "string"  }  },  "required": [  "message"  ]  } |
|  |  |

Table 18 – Tables: Add piece(s) to a table

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/tables /add | This endpoint is used to add a piece to a table. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "tableID": {  "type": "string"  }  "pieces": {  "type": "array",  "items": {  "type": “object”,  "properties": {  "pieceID": {  "type": "string"  },  "position": {  "type": "string"  }  }  }  }  },  "required": [  "requestorID",  "token",  "tableID",  "pieces"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  tableID: Identifier of an encrypted masking table  pieceID: identifier of a piece.  position: the position where the piece is located in the table. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The piece(s) have been successfully added to the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a set of data in JSON format. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "tableID": {  "type": "string"  },  "timestamp": {  "type": "string"  },  "pieces": {  "type": "array",  "items": {  "type": “object”,  "properties": {  "pieceID": {  "type": "string"  },  "position": {  "type": "string"  }  }  }  }  },  "required": [  "tableID",  "timestamp",  "pieces"  ]  } |
|  |  |

Table 19 – Tables: Delete a piece from a table

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/tables /{tableID}/deletepiece | This endpoint is used to delete a piece from a table. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "tableID": {  "type": "string"  }  "pieceID": {  "type": "string"  }  }  },  "required": [  "requestorID",  "token",  "tableID",  "pieceID"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  tableID: Identifier of an encrypted masking table.  pieceID: identifier of a piece. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The piece has been successfully deleted from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a message in JSON format. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "message": {  "type": "string"  }  },  "required": [  "message"  ]  } |
|  |  |

1. **Pieces**

Table 20 – Pieces: Register a piece

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/pieces | This endpoint is used to register a piece. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "addresses": {  "type": "string"  },  "tableID": {  "type": "string"  },  "position": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "addresses",  "tableID",  “position”  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  addresses: the addresses where the piece is stored in cloud/server[[2]](#footnote-2).  tableID: Identifier of the encrypted masking table that the piece relates to.  position: the position where the piece locates in the table. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The piece has been successfully created in the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains an index in JSON format signifying the index of the registered piece. The index is required to retrieve this particular piece. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "pieceID": {  "type": "string"  }  },  "required": [  "pieceID"  ]  } |
|  | Explanation | pieceID: identifier of a piece |

Table 21 – Pieces: Read a piece

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/pieces /read | This endpoint is used to return a piece by a given ID. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "pieceID": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "pieceID"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  pieceID: identifier of a piece. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The piece has been successfully read from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a set of data retrieved using the index in JSON format. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "pieceID": {  "type": "string"  },  "addresses": {  "type": "string"  },  "tableID": {  "type": "string"  },  "position": {  "type": "string"  }  }  "required": [  "pieceID",  "addresses",  "tableID",  "position"  ]  } |
|  |  |

Table 22 – Pieces: Delete a piece

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/pieces/delete | This endpoint is used to destroy a piece by its unique ID. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "pieceID": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "pieceID"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  pieceID: identifier of a piece. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The piece has been successfully deleted from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a message in JSON format signifying that the piece has been deleted successfully. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "message": {  "type": "string"  }  },  "required": [  "message"  ]  } |
|  |  |

The “addresses” need to be updated when an involving cloud/server goes offline. This is to ensure the availability of pieces.

Table 23 – Pieces: Update address in a piece

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/datamasking/pieces/update | This endpoint is used to update addresses in a piece. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "pieceID": {  "type": "string"  },  "addresses": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "pieceID",  "addresses"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  pieceID: identifier of a piece.  addresses: the new addresses where the piece and its replicas are stored in cloud/serves. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The piece has been successfully updated in the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a set of data retrieved using the index in JSON format. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "pieceID": {  "type": "string"  },  "addresses": {  "type": "string"  },  "tableID": {  "type": "string"  },  "position": {  "type": "string"  }  }  "required": [  "pieceID",  "addresses",  "tableID",  "position"  ]  } |
|  |  |  |

* 1. Support for anonymization

As described in the blockchain proposal, data sharing events will be stored in the blockchain. This helps data providers track data-sharing history in the federation, and provides integrity and availability of such history. We envision that the API supports two operations – store and read – exposed by two endpoints. This is to ensure that no one can alter the data once it has been logged. The API specifications for these two endpoints are presented below.

Table 24 – Keys: Register a data-sharing event

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/anonymization | This endpoint is used to register a data-sharing event. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "data\_provider": {  "type": "string"  },  "data\_consumer": {  "type": "string"  },  "timestamp": {  "type": "string"  },  "dataID ": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "data\_provider",  "data\_consumber",  "timestamp",  "dataID"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  data\_provider: name of the data provider who shares a dataset.  data\_consumer: name of the data consumer who receives a dataset.  dataID: Identifier of the shared dataset.  timestamp: The timestamp in the following format - "2006-01-02 15:04:05" |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data along with the parameters have been successfully stored in the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains an index in JSON format. The index is required to retrieve this event. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "index": {  "type": "integer"  }  },  "required": [  "index"  ]  } |
|  |  |  |

Table 25 – Keys: Read a data-sharing event

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | |
| /api/anonymization/read | This endpoint is used to retrieve a data-sharing event given the name data consumer. | |
| *Method* | POST | |
| *Consumes* | JSON |  |
| *Produces* | JSON |  |
| *Request parameters* | Body | Parameters submitted in JSON format. The names of the parameters and their types are specified in the following schema. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "requestorID": {  "type": "string"  },  "token": {  "type": "string"  },  "data\_consumer": {  "type": "string"  }  },  "required": [  "requestorID",  "token",  "data\_consumer"  ]  } |
| Parameter explanation | requestorID: Identifier of the requesting entity.  token: A crypto token to validate if the entity is allowed to perform the requested operation.  data\_consumer: name of the data consumer who receives a dataset. |
| *Response parameters* | HTTP status | Describes the status of the operation:  **200:** The data has been successfully read from the blockchain via the smart-contract.  **400:** Invalid request, required parameter(s) missing.  **401:** The operation is not allowed (unauthorised access, the token is invalid, etc.). |
| Body | The response body for a successful response (response code 200) contains a set of data retrieved using the index in JSON format. |
| Schema | {  "$schema": "http://json-schema.org/draft-04/schema#",  "type": "object",  "properties": {  "data\_provider": {  "type": "string"  },  "data\_consumer": {  "type": "string"  },  "timestamp": {  "type": "string"  }  "dataID": {  "type": "string"  }  },  "required": [  "data\_provider",  "data\_consumer",  "timestamp",  "dataID"  ]  } |
|  |  |

1. **Other functionalities**

The particulars of other functionalities have not been formalised yet. This API will be updated accordingly once the relevant information for a functionality has been agreed upon by the respective partners.

1. **Comments**

* The SLA API can be specified in the similar way as the specification of the policy storage API.
* We have used the concept of a crypto token without specifying it. This is something that needs to be decided by WG partners. One suggestion: we could leverage PKI by using X.509 attribute certificates for this.

1. We will use “user” to represent “authorised data consumer” in the following sections. [↑](#footnote-ref-1)
2. The “addresses” contains all the addresses that the piece and its replicas are stored. For example, if the number of replicas is 2, then the value of “addresses” is address1+address2+address3. [↑](#footnote-ref-2)