Content

Client Protocol

This chapter is describing the protocol for using the vaccinator pseudonymisation service. This is the internal communication for the vaccinator service and not to become implemented by customers developers.

The client protocol is implemented in JavaScipt and offers access to all pseudonymisation functions and management functions of vaccinator. **It is not the endpoint presented to the developers.** Instead, this is communication between the JavaScript class and the vaccinator service itself.

The endpoint interface for developers is described in the next chapter "Client API".

Implementation details

The protocol is REST based and works by sending information using POST requests. The functionality at the service provider is established by some sort of intermediate handling between client and identity management. The described client protocol is used between the client implementation and the vaccinator service. The service provider always forwards all the requests to the vaccinator service and only adds his access information fields to the request (sid and spwd). The results are also forwarded back to the calling client then. A few functions like pAdd also return data that is interesting for the service provider. He can use this to update his own database.

The JSON encoded structure, containing the function name and all needed parameters, is typically sent in a POST field named `json`. A typical call consists of some operation code (op) and some field containing the encrypted data for the vaccinator service (`data`). The optional `uid` field is for API users to identify calls to some specific user or assigning return values to internal identifiers. It can be unset (empty) or contain some value. It is not used for vaccinator identity management.

All calls will return a JSON encoded string, containing the result of the operation and additional information.

There is always a `status` field returned. If it is not OK, something went wrong. In this case, the status is either INVALID or ERROR. An additional `code` field contains the error number and the `desc` field contains additional information about the error (only if `status` is not OK).

The `data` field contains encrypted payload for the identity management. It is encrypted due to contained receipt. A data field is always encoded like this:

```
receipt:cs:iv:payload
```

- The `receipt` defines the used algorithm for encryption.
- The `cs` value is the checksum of the used app-id key (last byte in hex, see app-id description).
- The `iv` is the start vector/nonce for encryption in hex encoding.
- The `payload` is the hex or base64 encoded JSON string with encrypted payload data.

It typically looks like in this example:

```
chacha20:f7:29a1c8b68d8a:Z2Zkc2dmZG1rZyBmZ(...)XI0N2Z2IDZyNHkzMmY0Mw==
```

This encryption is done automatically by the client API and happens transparently for the end users and service provider developers.

NOTE: By this encryption, using the app-id as key, the service provider and the vaccinator service both do not have access to the content (for example patient data). The checksum as part of the receipt allows later verification, if the dataset was encrypted with one or maybe a newer app-id. This is useful if, for example, the changeAppId() function failed during processing (please refer to changeAppId() function description in API description).

NOTE: The above chosen ChaCha20/12 cipher is just the reference implementation. You can also use AES 256 (cbc-aes-256) or other encryption algorithms. The only thing to respect is the encoding using hex for the `cs` and the `iv/nonce`.

References:

https://en.wikipedia.org/wiki/Salsa20#ChaCha20_adoption

Transport encryption

Of course, all API REST calls are using standard SSL connections (https). But in order to make sure that even the service provider does not know the end users identities, some data is also encrypted. The encrypted information is transported in the `data` field of the POST calls.

The `data` field is encrypted using the SHA256 from the end users app-id as password. We will start implementing ChaCha20 encryption with individual IV. See above chapter about implementation details.

Error codes

In case of an error, the `status` value is not OK, instead it is either INVALID or ERROR. INVALID means that some data you provided triggered the error and ERROR is some vaccinator related internal error. If INVALID, you need to check your input.

The system then returns two additional fields:

code	desc	status
1	Missing Parameters.	INVALID
2	Wrong Protocol.	INVALID
3	Your software seems outdated.	INVALID
4	The account was locked due to possible misuse.	INVALID
5	Invalid credentials (check sid and spwd).	INVALID
6	Invalid encoding (check data values and JSON integrity).	INVALID
7	Not found (pid is not found in the system).	INVALID
8	Invalid partner (you are not allowed to access foreign data).	INVALID
9	Invalid parameter size (some parameter exceeds limits).	INVALID
99	Some internal service error happened. Please contact %PRODUCT% support.	ERROR

Add new person

This call is adding a new person to the system.

Field	Description
op	add
data	Encrypted data for the containing all the data to be stored (string blob, use base64 encoding for binary data). Please follow the encoding scheme described in "Implementation Details".
uid	User identifier provided by the API user.

Result:

Field	Description
status	Either OK, INVALID or ERROR. See generic description for details.
uid	User identifier provided by the API user during call (only if it was provided).
pid	New id for the newly generated person. This may be stored by the service provider and get assigned to the calling client (identified by uid).

Important implementation note: If you forward some positive result to the client, please take the returned `pid` and add this to your service provider database while assigning to the user. By this, you are able to send your client software a complete and up to date list of all PIDs at any time.

Update person

This call is updating an existing entry.

Field	Description
op	update
data	Encrypted payload containing all the data to get updated (string blob, use b64 encoding for binary data).
pid	Person ID to update.
uid	User identifier provided by the API user.

Result:

Field	Description
status	Either OK, INVALID or ERROR. See generic description for details.
uid	User identifier provided by the API user during call (only if it was provided).

Important implementation note: Updating payload data is critical to the local caches of the JS class. If multiple systems accessing the data, the cache of the other systems is outdated after some update. Only the system which did the changes is up to date. Therefore, this has to be handled special: Please create a unique code (eg time stamp or random number) in case you forward some `update` request to the vaccinator service. This code has to be sent to your client application as soon as possible (maybe as part of your protocol). There, please call the `wipeCache()` function with this code every time. This will trigger the local cache to refresh in case something has changed. Please refer to the `wipeCache()` function description in API documentation.

Retrieve person

This call is retrieving the data of one or more existing entries.

Field	Description
op	get
pid	Person ID to retrieve data from. Multiple PIDs may become requested by concatenating them using blank as divider character. The maximum allowed PIDs is 500 per request.
uid	User identifier provided by the API user.

Result:

Field	Description
status	Either OK, INVALID or ERROR. See generic description for details.
uid	User identifier provided by the API user during call (only if it was provided).
data	This contains the payload(s). Payload always comes as a object array where the PID is the key. It has one entry in case only one PID was requested and multiple entries in case of multiple results. Every given PID creates a return value, even if it was not found or suspicious. Note: The order is not guaranteed to be the same as provided in the request!

The returned result always confirms to this JSON schema, written as a complete example answer:

```
{
  "status": "OK",
  "version": "0.0.0.0",
  "uid": 12345,
  "data": {
    "f315db7b01721026308a5346ce3cb513": {
      "status": "OK",
      "data":
"chacha20:7f:29a1c8b68d8a:btewwyzox3i3fe4cg6a1qzi8pqoqa55orzf4bcxtjfcf5chep998sj6"
    },
    "2ff18992cfc290d3d648aea5bdea38b1": {
      "status": "NOTFOUND",
      "data": false
    }
  }
}
```

The above example showing the result of a request with two PIDs. The first was a valid request, the second was some unknown entry.

Delete person

This call is deleting an existing entry.

Field	Description
op	delete
pid	Person ID to delete from service. Multiple PIDs may become requested by concatenating them using blank as divider character. The maximum allowed PIDs is 500 per request.
uid	User identifier provided by the API user.

Result:

Field	Description
status	Either OK, INVALID or ERROR. See generic description for details.
uid	User identifier provided by the API user during call (only if it was provided).

Check connection

This is just a simple "ping" sort of call to verify if the service is available.

Field	Description
op	check
uid	User identifier provided by the API user.

Result:

Field	Description
status	OK
uid	User identifier provided by the API user during call (only if it was provided).

This is not a function that does anything. It is just answering with status "OK". This is also not verifying the validity using `sid` and `spwd`.

Implementation of protocol forward

This chapter explains, what a service provider has to do to successfully handle and forward REST protocol requests.

Forward requests by adding service provider credentials

In general, all requests have to become forwarded to the vaccinator service URL. The JSON encoded in `json` data value must get enhanced by two additional values:

Field	Description
sid	The service provider ID. This is provided to the service provider by the vaccinator service staff.
spwd	The service provider password. This is provided to the service provider by the vaccinator service staff.

Upon the JSON request was updated by sid and spwd, the request is forwarded to the vaccinator service URL (provided to you by the vaccinator service staff). The returned result is sent as an answer to the calling end user client (eg web browser API).

Observe and enrich function calls

In addition, the service provider has to observe the functions to provide additional functionality required.

update call

The update call will out date all other participants local caches. Therefore, they need to know about this. The only party able to tell them is you. This is done by acting in case of a positive "update" call. In case the vaccinator service announces success, please generate a time stamp (or random token) and provide it to all affected clients. By knowing the pid from the request, you should be able to know the affected persons (logins). You send them this time stamp with their next request and they will have to call the wipeCache() function with this as parameter. If the API recognises this time stamp/token as already know, nothing will happen. If it does not know this value yet, it will wipe it's cache and regenerate it on demand later.

ALL user requests (get, update and delete)

Here you might want to verify it the logged in user is allowed to handle data about this PID. This would be some important security layer to prevent manipulations in local client to retrieve or manipulate data of PIDs the user is not allowed to. Here, please forward the request only if the user is allowed to. Please follow the protocol description above and, if not allowed, send some status "INVALID" and code 7 (pid not found).