eHealth Network Guidelines

to

the EU Member States and the European Commission

on

Interoperability specifications for cross-border transmission chains between approved apps

Detailed interoperability elements

between COVID+ Keys driven solutions

V1.0

2020-06-16

The eHealth Network is a voluntary network, set up under article 14 of Directive 2011/24/EU.
It provides a platform of Member States' competent authorities dealing with eHealth.

Adopted by consensus by the eHealth Network working group on COVID+ keys driven solutions for tracing apps

European Proximity Tracing

An Interoperability Architecture

The least complex and most robust way to connect the backends behind all the different national proximity tracing apps is a **Federation Gateway Service**, which accepts diagnosis keys from all countries, buffers them temporarily, and provides them for all countries to be downloaded. Additionally, all backends can be informed immediately if new data is available, so that transmission lags are kept minimal. In this document, we propose a definite ready-to-implement architecture of the Federation Gateway Service.

IMPRINT

Table 1: Imprint Contact

Issued by	T-Systems/SAP

Title
European Proximity Tracing—An Interoperability Architecture

Version	Last Review	Status
1.0	16.06.2020	Released

Authors	E-Mail
, T-Systems	martin.mitev@t-systems.com
Wolfram Plettscher, T-Systems	wolfram.plettscher@t-systems.com
Hannes Rollin, T-Systems	hannes.rollin@t-systems.com
Steffen Schulze, T-Systems	steffen.schulze@t-systems.com

Contributors	E-Mail
Dr. Marek Meyer, Arago	mmeyer@arago.co
Lukas Masuch, SAP	lukas.masuch@sap.com

Reviewers	E-Mail
Dirk-Willem van Gulik	dirkx@webweaving.org
Cas Cremers, CISPA	cremers@cispa.saarland
Niall Sinnott	niall_sinnott@health.gov.ie
Rainer Schweigkoffer, SAP	rainer.schweigkoffer@sap.com
Mirko Sitte, Telekom	mirko.sitte@telekom.de

TABLE OF CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

1 Introduction

1.1 Context

Most European countries are developing proximity tracing apps to reduce the spreading of COVID-19, generally using the Exposure Notifications API from Google and Apple. While the proximity detection mechanisms of these apps are compatible, the national backends behind the different national apps—as yet—don't talk to each other. This is unfortunate, as Europeans commute and travel all over the continent; interoperability of the national backends is a must.

Several interoperability patterns have been discussed in the document titled "European Interoperability—Conceptual View".

The pattern preferred by the European eHealth Network is a **single European Federation Gateway Service**. Each national backend uploads the keys of newly infected citizens ("diagnosis keys") every couple of hours and downloads the diagnosis keys from the other countries participating in this scheme. That's it. Data conversion and filtering is done in the national backends.

This document is an architectural specification of the Federation Gateway Service, comprising its general functioning, interface specification, data structures, security aspects, traffic volume estimates, and storage options.
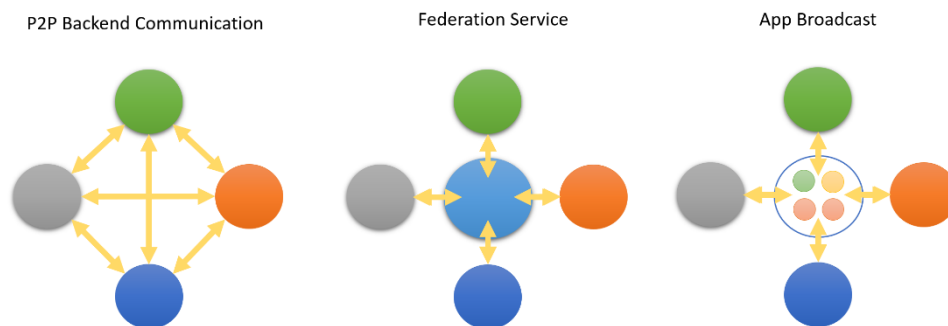
Figure 1: Comparison of interoperability patterns

1.2 Scope of Document

From a TOGAF methodology point of view, we mainly cover the aspects of

- "C. Information System Architectures" and
- "D. Technology Architecture"

related to the Federation Gateway Service.
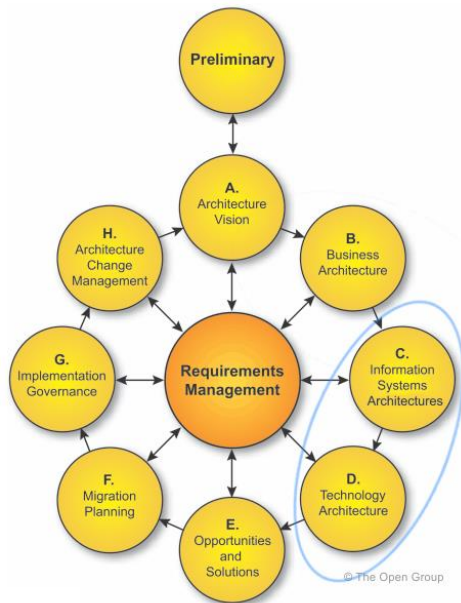
Figure 2: TOGAF Architecture Model

2   Architecture Overview

As said before, the Federation Gateway Service is used to synchronize the diagnosis keys across all national backend servers.

The amount of data uploaded by each backend server is comparatively miniscule; we're talking about 20-30 MB per day at most (compare section 5.5). Additionally, the number of participants is restricted, since each country operates only one backend. It follows that a small web service, equipped with a simple load balancer and replicated storage to ensure high availability, is enough to meet the demand in even the most unwelcome pandemic scenarios.

The following figure gives an overview of the Federation Gateway Service as specified in this document:
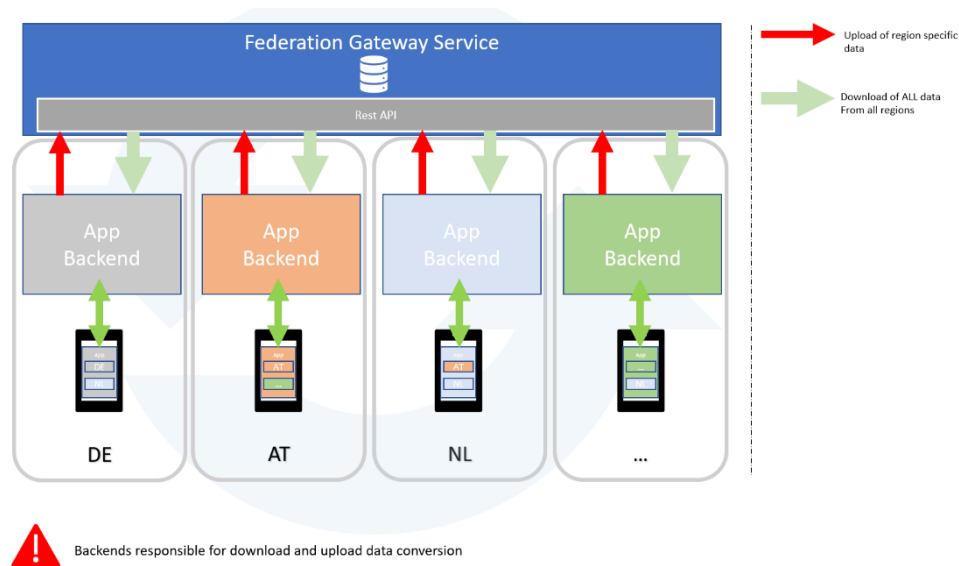
Figure 3: Federation Gateway Service Overview

By using the Federation Gateway Service, backend-to-backend integration is facilitated and countries can onboard incrementally, while the national backends retain flexibility and control over data distribution to their users.
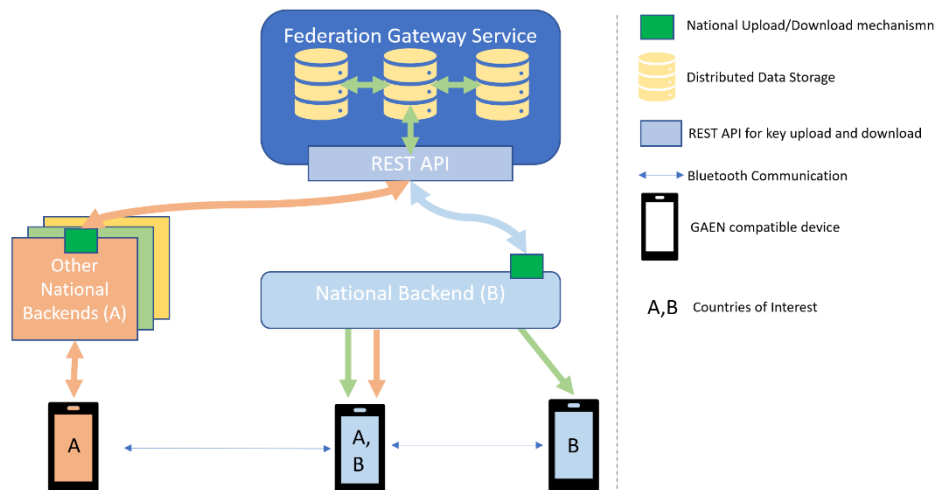
Figure 4: Autonomous National Backends

As seen in figure 4, each device communicates *only* with the corresponding national backend. In this case, the app user to the left (say, Alice from country A) has received a positive test result, so she submits her diagnosis key to her backend. The diagnosis key is then uploaded to the Federation Gateway Service, downloaded by the backend of country B, and finally downloaded by those users in country B who traveled to country B. Only those who had close contact with Alice, however, will be warned of possible exposure.

2.1 Approach

We're advocating a **Federation Gateway Service**, where *all* participating national backends upload *all* diagnostic keys received from their respective users, and each participating backend downloads *all* diagnostic keys from *all* other countries. It might be the case that some countries generally don't accept certain countries or would like to reject diagnosis keys that have certain characteristics. Nevertheless, the Federation Gateway Service always provides everything, and the national backends may filter the data according to their needs.

In a nutshell, the Federation Gateway Service stores the information of currently infected citizens plus the countries they visited ("countries of interest"), but it doesn't know the true identity of the citizens, and it doesn't know who came into close proximity of the infected citizens. Healthy but exposed citizens need all diagnosis keys from all their countries of interest, since the matching of diagnosis keys to exposure data *happens on the mobile devices.* Not even the national backends have access to that information to prevent contact tracking.

Naturally, all users need to specify their countries of interest correctly, either manually or automatically. Only then the whole fleet of European proximity detection apps is truly interoperable.

2.2 Assumptions

The main **assumptions** of this architecture are the following:

- Data volume of **new** diagnosis keys per country and day is typically up to 10-20 MB. As an upper bound the volume can therefore be estimated as less than 1 GB per day and country.
- Data is transferred batch-wise every few hours, **not in real-time**
- Google/Apple Exposure Notification API **(GAEN)** is used by all participating countries
- **Diagnosis key** information uses GAEN format, including visited countries ("countries of interest") for each key
- Countries may process, distribute and publish diagnosis keys. If diagnosis keys are considered PII according to GDPR (legal review pending), the issuer of each national app will ensure compliance with GDPR.
- Citizen are using the app of their home country
- National apps communicate only with the corresponding national backend

3   Communication

3.1.1   Device-to-Device Communication

All apps using the [Exposure Notification API](#) (EN) by Google and Apple for proximity detection are compatible. Fortunately, most European countries have subscribed to this approach. If two citizens, no matter where they are from, are using EN-enabled apps, the EN mechanism detects proximity and duration of contact in a non-traceable manner on both devices via a modified Bluetooth handshake.

> ⓘ The Exposure Notification API at this point of time does not support the exchange of country codes. Moreover, such a feature is generally not endorsed, as it could be abused to build "foreigner scanners."

The countries of interest—or countries visited—have to be determined by the app, using either mobile provider metadata or manual user entries.

However, if there are two citizens from different states, so that at least one of them does not use an EN-enabled app, proximity detection for them is as yet impossible.
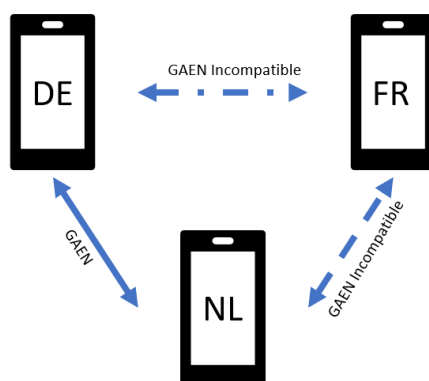
Figure 5: Device to Device Communication

3.1.2   Device-to-Backend Communication

Exactly how each national app communicates with the corresponding national backend— whether via CDN, active push, or otherwise—is completely left to each country, as long as the GAEN requirements are met. The beauty of the Federation Gateway Service is that it doesn't restrict the national apps in any way except one: The exchange format is specified.
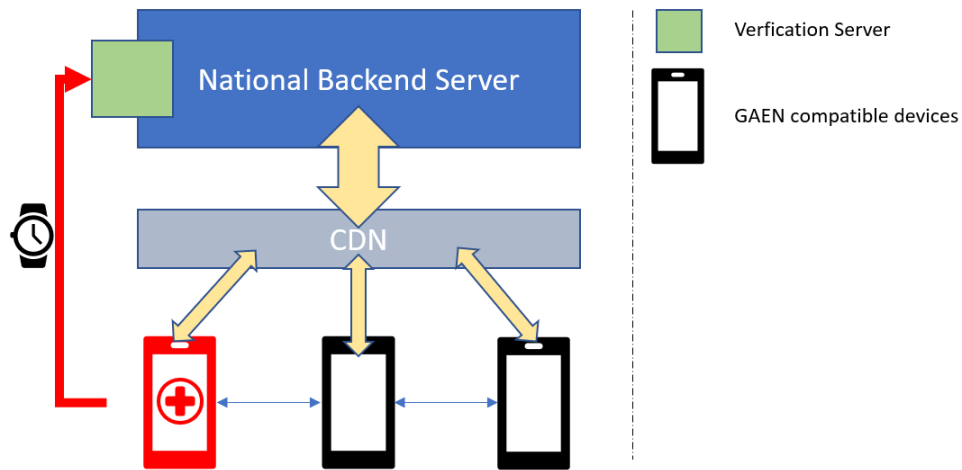
Figure 6: Example for Device-to-Backend Communication

3.1.3 Backend-to-Backend Communication

A direct backend-to-backend communication is not necessary, because the main purpose of the Federation Gateway Service solution is to provide the new diagnosis keys. All participating national backends will provide the new diagnosis keys of their citizens to the Federation Gateway Service, which in turn stores the keys and provides them for download. Nevertheless, bilateral communication between national backends is not categorically excluded—it's just not necessary for those countries that are connected to the Federation Gateway Service.
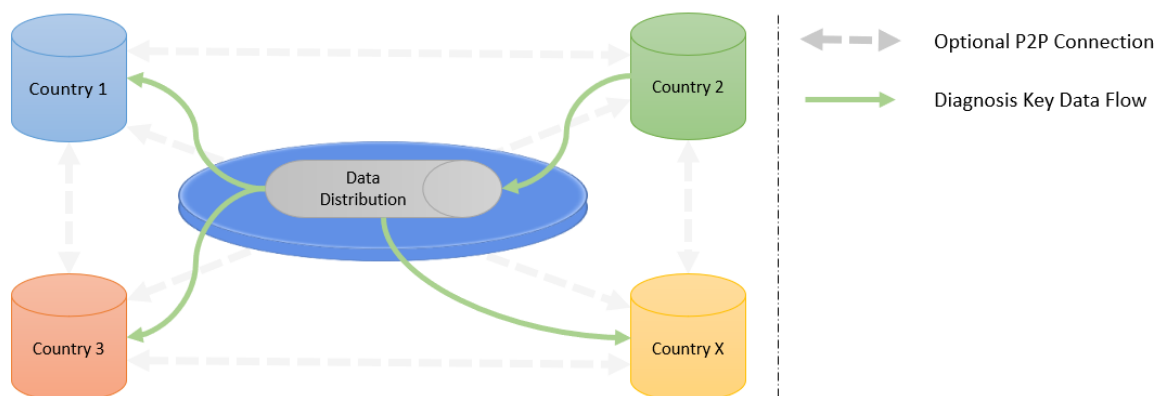
Figure 7: Indirect Backend-to-Backend Communication

As shown in the figure, uploaded data from one country is distributed to all other countries. Each national backend, then, stores all diagnosis keys of all other countries and can provide the keys, filtered by countries of interest, to their own users.

i   The Federation Gateway Service is a slightly different from a Forwarding Gateway, because the data is temporarily stored by the Federation Gateway Service for retrieval and not actively forwarded. The main reason for buffering the data is this: Directly forwarded data may get lost if the receiver is not available, which is likely to happen at least occasionally. Passively provided data can be downloaded by the backends at their convenience.

i   A VPN connection is optional, because we already have encryption in transit via TLS.

4 Data Structures

4.1 Data Types

4.1.1 Google Exposure Notification Keys

All diagnosis keys are based on the GAEN Key Export File Format in Version 1.4 described here:

https://static.googleusercontent.com/media/www.google.com/de//covid19/exposurenotifications/pdfs/Exposure-Key-File-Format-and-Verification.pdf

The GAEN key signature is ignored and replaced by a PKI signature in the exchange format (more details in section 4.1.6).

⚠️   The exports need to be generated and signed by each national backend.

4.1.2 Country Codes

All country codes are based on ISO 3166-1:

https://www.unece.org/cefact/locode/service/location

In this specification is an option to specify the location as well:

https://www.unece.org/fileadmin/DAM/cefact/locode/de.htm

Examples:

LOCODE Representation	Translation
DE	Germany
ES	Spain
IT	Italy
NL	Netherlands

Table 2: Country Code Representation

ℹ️   Google and Apple are increasingly using the Mobile Country Code as region/country identifier in their documentations, which has to be considered in a backend implementation.

4.1.3 Transmission Data Type

In addition to the diagnosis key, each user has to transmit the visited countries (countries of interest) to the national backend. Example:

```
message SubmissionPayload {

  repeated Key keys = 1;
  repeated string regions = 2;
}

message Key {

  bytes keyData = 1; // key of infected user
  uint32 rollingStartIntervalNumber = 2;
  uint32 rollingPeriod = 3; // number of 10-minute windows between key rolling
  int32 transmissionRiskLevel = 4; // risk of transmission
}
```

Figure 8: Example Submission Payload for Diagnosis Keys (App-to-Backend)

With this information, each national backend can transfer exchange information for diagnosis keys with a verification type and an origin country, key by key in a batch to the Federation Gateway Service:

```
message DiagnosisKey {

  bytes keyData = 1; // key
  uint32 rollingStartIntervalNumber = 2;
  uint32 rollingPeriod = 3; // number of 10-minute windows between key rolling
  int32 transmissionRiskLevel = 4; // risk of transmission
  repeated string visitedCountries = 5;
  string origin = 6; // country of origin
  VerificationType verficationType = 7; // set by backend
}

enum VerificationType {

  LAB_VERIFIED = 0,
  SELF_REPORTED = 1,
  ...
}
```

```
message DiagnosisKeyBatch{
```

```
  repeated DiagnosisKey keys = 1;
}
```

Figure 9: Diagnosis Key Payload (Backend-to-Federation Gateway Service)

⚠ The verification type must be defined in more detail for a European-wide standardized solution.

⚠ The values Verification Type and Origin are set by the national backend; Origin is necessary to know where the data is coming from during the download. All other values can be mapped from the App input.

ⓘ At this time the 'transmissionRiskLevel' parameter is not yet supported. Member states may—as a compensating measure and from a GDRP perspective—elect to set this value to 0x7FFFFFFF to reduce the risk of data leakage and misinterpretation.

4.1.4 Client Certificates

The identity of an uploading instance is derived from an X.509 certificate issued by appropriate authority. This certificate contains country, location, common name, and other values which can be used in the architecture for security and identification purposes.

4.1.5 Hash Calculation

For a correct SHA256 hash calculation across different programming languages and data formats, it's important to use the same pattern for extracting the bytes to be used in the hash function. This ensures to get the exact hash independently of format (XML, JSON or protobuf) in every programming language.

⚠️ Hash calculation over the raw content is not recommended because a lot of different frameworks can disturb the calculation. The calculation should be done after serialization.

4.1.6 Signature

Signatures are created in the PKC7 Standard to use the advantages of an Public Key Infrastructure like Certificate Revocation, Rollover etc. This Cryptographic Message Standard is defined in RFC5652 (https://tools.ietf.org/html/rfc5652). These signatures are created from the hashed data content and certificate information, for later usage in Base64 format to describe the content of an uploaded batch described in RFC4648. (https://tools.ietf.org/html/rfc4648).

4.2 Data Storage

Uploaded diagnosis keys are stored for 14 days. While theoretically unnecessary if direct forwarding is used, practical considerations make temporary buffering worthwhile:

1. Packets get lost and backends may be unavailable. With stored data, download retries are possible.
2. Timing of downloads is left to the backends instead of forcing a schedule.
3. Newly onboarded countries get the data for the past 14 days at once, so they don't miss important data.

ℹ️ Since newly infected citizens initially submit up to 14 daily keys, stored keys can be up to 28 days old.

4.2.1 Database Requirements

As for database technology, we gathered the following requirements:

Requirement	Explanation
Object Storage	The database must support storage of different objects without needing schema changes
Strong Consistency	The database must support strong consistency, i.e., new data is fully replicated after each transaction
Data Expiry	All stored diagnosis keys have a lifetime of 14 days
Download by Date	National backend wants to download only new data or data newer than a specific date
High Availability	We need redundant, replicated storage to avoid down-time
Medium Scalability	If other countries join, the systems need to scale out to provide fast uploads and downloads
Secondary Indices	The diagnosis keys need to be classified by multiple arguments, e.g., by timestamp and diagnosis type

Table 3: Data Storage Requirements

Regarding these requirements, the preferred database is a document-oriented database.

ⓘ According to the CAP theorem for distributed data storage, only two of the three requirements *consistency*, *availability*, and *partition tolerance* can be fully met at the same time. Partition tolerance refers to resilience against message loss across the network. Since consistency and availability provide the greatest value for the national backends—and since both the number of partitions and message loss rate will be small—we focus on the first two requirements.

4.2.2 Database

Related to the requirements, a document-oriented NoSQL DB is used to ensure the compatibility between all current and future formats.

4.2.3 Database Structure

The database structure should provide multiple read nodes to avoid performance gaps, especially since download traffic is much higher than upload traffic—to be precise, download traffic is about n times higher, if n is the number of participating countries. Moreover, high availability requirements imply replication across at least two geographically separate regions.
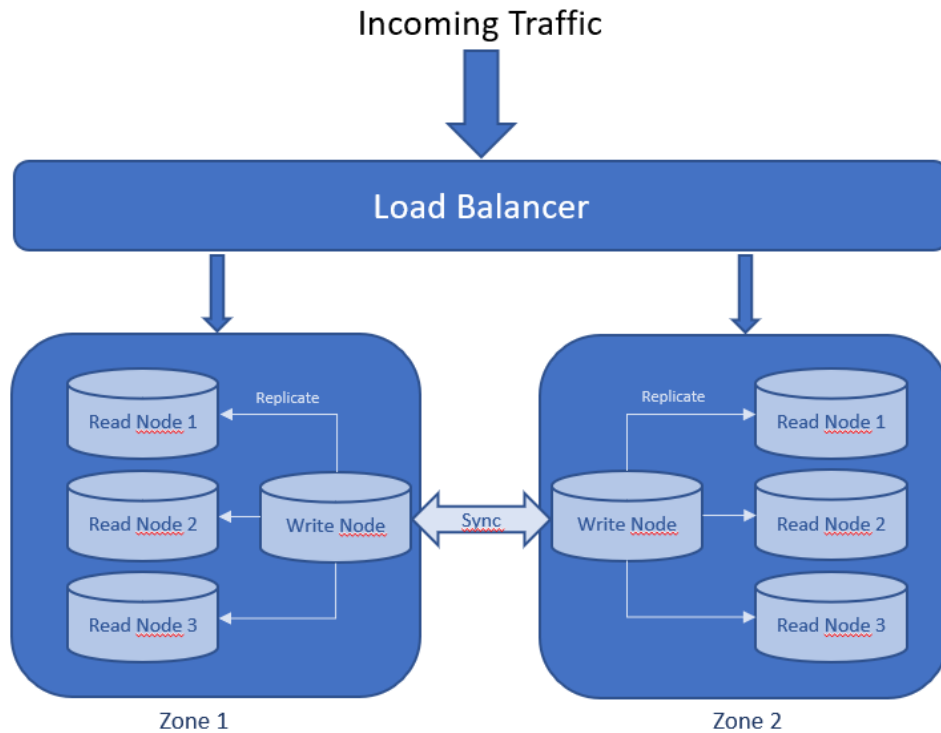
Incoming Traffic

Figure 10: Database structure; API and outgoing traffic omitted for clarity

i The exact mechanism for data synchronization is an implementation detail depending on the concrete database technology and its configuration.

4.2.4 Data Format

A document in the database needs the uploader metadata, a payload, a flag "diagnosis type" (to differentiate between self-diagnosis, lab-tested, and others), format information, and a batch tag related to the upload. The document itself represents a single diagnosis key together with uploader, format and batch information:

```
{
    "_id": "string",
    "inserted": "timestamp",
    "batchTag": "object",
    "uploader": {
            "batchTag": "string"
            "batchSignature": "string" // signature of entire upload batch
            "thumbprint": "string",
            "commonName": "string",
            "country": "string"
            // ...more certificate information...
        },
    "format":     {
            "format": "string",
            "version": "string"
        },
    "payloadHash": "string" // payload hash (e.g., SHA256)
    "payload": "object" // type: DiagnosisKey Payload
```

```
    }
```

To ensure compatibility, the payload is described by a format information which indicates the type and object version used. This is necessary to ensure compatibility with different formats.

i   Metadata of the uploader is extracted from client certificate. This includes common name, country, thumbprint and other certificate details.

i   The payload hash is an SHA256 representation of the payload. This hash is used to ensure the uniqueness of each diagnosis key within the database.

4.2.5   Document Size

Document size is small, since each diagnosis key is stored in a single document. When a batch of diagnosis keys is received, the API stores each key set as a single small document. This avoids query performance gaps, ensures flexibility, and makes it easier to query the data. Of course, some redundancy has to be accepted.

4.2.6   Document Expiry

The documents expire automatically after 14 days.

4.2.7   Secondary Index

For effective querying, secondary indices for uploader country and diagnosis type are necessary.

4.2.8   Document Batching

The documents need to be split into batches to minimize download problems. During upload, the Federation Gateway Service bundles incoming documents into batches of a fixed size, e.g., 5000 diagnosis keys per batch, so that downloads are split into bite-sized chunks—the batches—by design. After upload completion, the documents are marked with a unique batch tag.
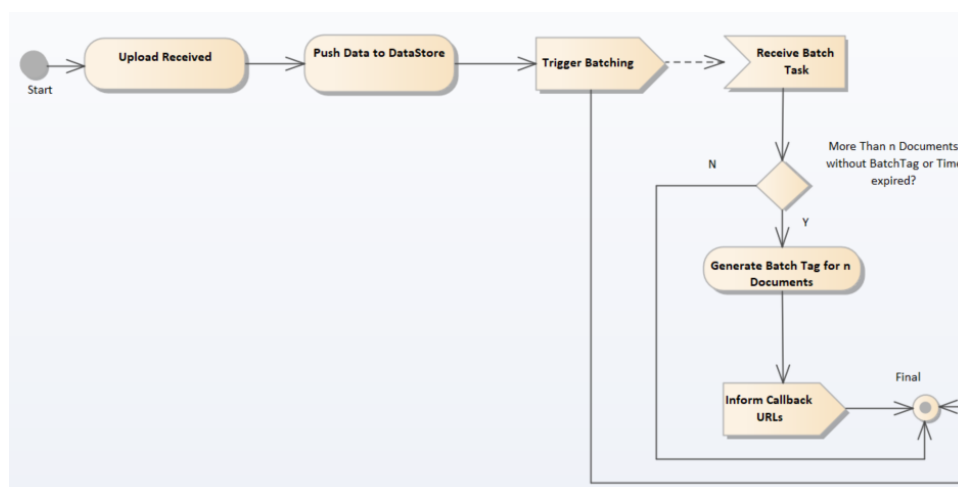
Figure 11: Batching Process

4.2.9 Document Batch Tag

As seen in the data format, the data storage documents have two batch tags, one in the uploader section and one in the root document.

Here's why: The uploader tag is used to identify the documents of the uploader. The other tag is used to identify the documents across *all* uploaders, which is important during the download. Therefore, both of them have a different data type. The uploader tag is an arbitrary unique value provided by the uploader. The other tag is an object which needs to be incremental and unique per day, because it's used to "navigate" within the day.

Example from MongoDB: https://docs.mongodb.com/manual/reference/method/ObjectId/

Definition	Value
4 Byte	Timestamp
5 Byte	Random Value
3 Byte	Incrementing Value

Table 4: MongoDB ObjectId Definition

This definition results in a globally unique and incremental hexadecimal string, the Object ID, which will be used as batchTag.

> A timestamp for navigation within the day is not recommended, because it's very hard to hit the "right second" in a data query, if a format like 01-22-2020-20:20:20:43434Z is used. A batchTag together with a date is much easier to handle in case of thousands of batches per day.

5 Interfaces

5.1 Overview

The Federation Gateway Service provides a simple REST API with four access points, one for update, one for download, one for callback registration, and one for auditing.
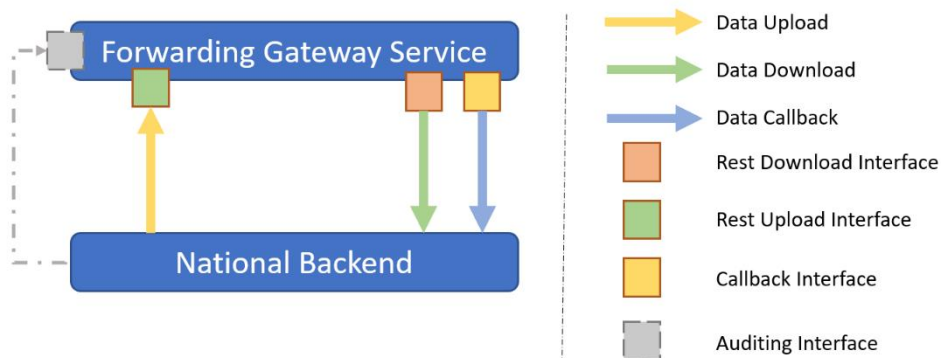
Figure 12: API Overview

Purpose of the interfaces are in a few words: download of diagnosis keys, upload diagnosis keys, get notified if new diagnosis keys are available and audit the system from outside.

For detailed description of REST interfaces, we rely on the Open API Specification 3.0. This allows a comprehensive human-readable and machine-readable representation of all aspects of the defined interface.

We defined three access points were which have the following scheme:
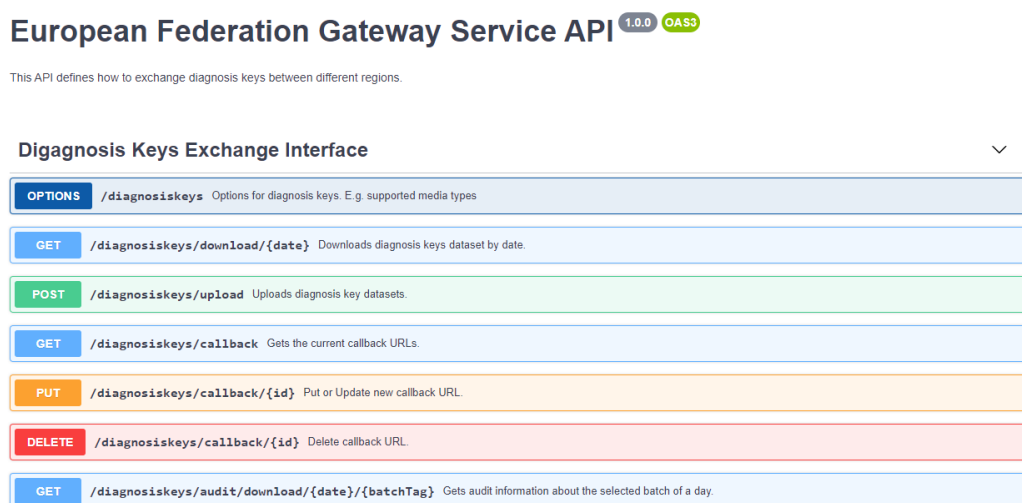
Figure 13: Open API Definition Overview

⚠️ The Federation Gateway Service API performs no signing of data packages according to GAEN specifications. Each national backend needs to pack and sign the data by itself.

5.2 Versioning

The REST API uses versioning within the Accept Header to negotiate content types. This ensures compatibility between different upload formats. The pattern for the Accept header is:

```
application/[MIME-SubType]+v[Version]
```

Examples:

```
application/json+v1.0

application/protobuf+v1.0
```

This format ensures the exact content for national backends and avoids API duplications and broken links because of mixed formats between different countries. The format of the version number is defined by semver (https://semver.org/). In a few words: the major version number is changing for incompatible API changes, the minor version for backwards compatible changes and patch version for bugfixes (optional).

ℹ️ The exact data format has to be negotiated between the member states.

⚠️ **Standard MIME types are not accepted.**

⚠️ Implicit conversion between major versions of data formats is not supported. Means: upload in v1.0 and download in v2.0 is not possible. Backwards compatibility is given within minor versions.

5.3 Download Interface

5.3.1 Overview

The download interface consists of one possible request for retrieving a batch of diagnosis keys.

```
GET  /diagnosiskeys/download/{date}  Downloads diagnosis key dataset by date.
```
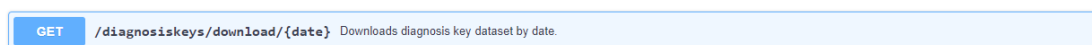
Figure 14: Download Interface

The request accepts only a date variable; this indicates the maximum age of requested diagnosis keys. In other words, only diagnosis keys newer than `{date}` will be downloaded.

⚠️ The download affects only diagnosis keys which are not uploaded by the requesting backend (verified by the client certificate identity information).

5.3.2 Parameters

Figure 15: Download Parameter Definition

Here's a brief explanation of the download batchTag:

If a download is triggered, there might be thousands of diagnosis keys available, so that the API returns just the first batch with a tag (see Response Codes). The same download call is then repeated, but including the received tag, so that the next batch is returned. This improves performance and fault tolerance.

⚠️   The download batchTag is unrelated to the upload batchTag.

5.3.3   Responses

Figure 16: Download Responses

5.3.4   Transmission Protocol

The download is triggered by calling the download URL with the timestamp of the last query. If the client certificate is valid and the requested content type is available, the data will be queried and transformed into the response.
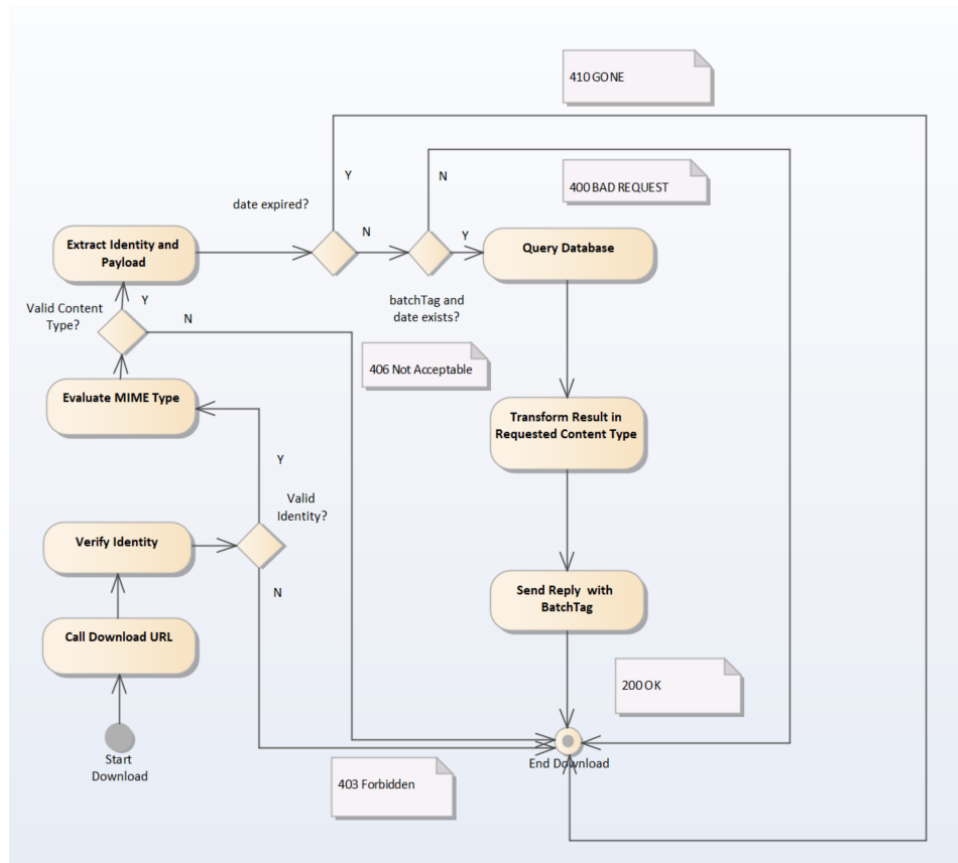
Figure 17: Download Transmission Flow

ⓘ To get all data, the download operation needs to be done multiple times, if the number of batches exceeds one. The last call is empty and returns the same timestamp as requested.

5.3.5   Client Process

The client process is defined as active polling:
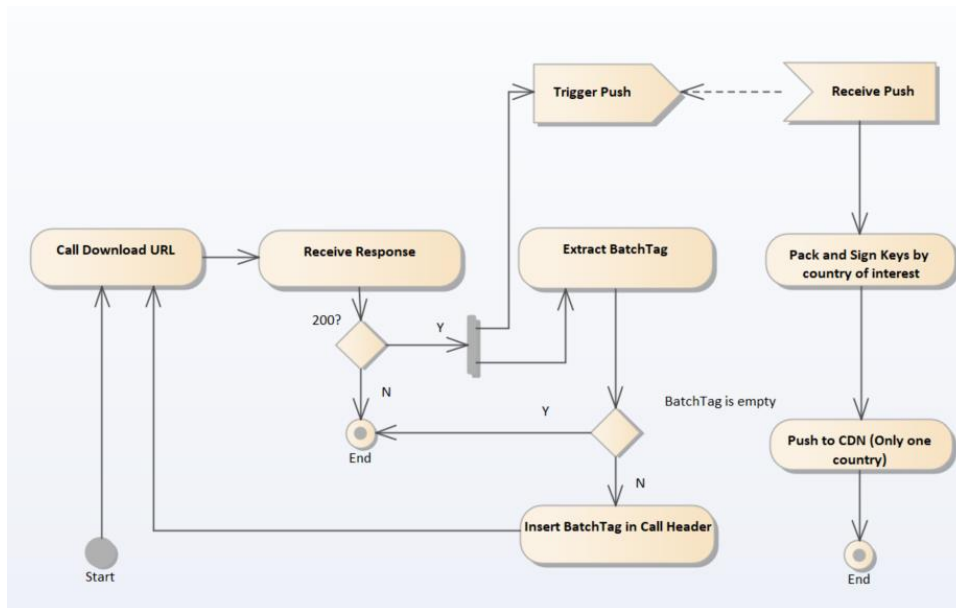
Figure 18: Download Client Process

⚠ Each national backend is responsible for packing and publishing keys for their own citizens. The implementations of the various national backends can be different.

5.4 Upload Interface

5.4.1 Overview

The upload interface consists of one call to upload a set of diagnosis keys, potentially separated into several batches:

Figure 19: Upload Interface

eHealth Network

5.4.2 Parameters

Name	Description
batchTag * required string (header)	Required Tag to tag the send batch (must be not unique). 123456
batchSignature * required string (header)	PKC7 Payload signature in Base64 encoding. ABDBJ3459MMDJ3223...

Figure 20: Upload Parameters

Here's a brief explanation of the upload batchTag:

If an upload is triggered, the Federation Gateway Service accepts a batchTag as a group identifier for uploaded payloads. This supports possible delete, update, and release actions in the future.

⚠ The batchTag in the download section is unrelated to the upload batchTag.

ⓘ The upload batchTag can be chosen arbitrarily. The API appends uploaded payloads to the same set and returns the submitted tag.

⚠ The batchSignature has to be calculated over the individual keys inside the batch instead of the batch itself.

5.4.3 Responses

Code	Description
200	OK.
201	Database Entries created.

Headers:

Name	Description	Type
batchTag		string *Example: 123456*

Code	Description
207	Data partially added with warnings. More details in document.

Media type

```
application/json+v1.0        ▼
```
Controls Accept header.

Example Value

```json
{
  "201": [
    1,
    2,
    5,
    8,
    9
  ],
  "409": [
    3,
    4,
    6,
    7
  ],
  "500": [
    10
  ]
}
```

Headers:

Name	Description	Type
batchTag		string *Example: 123456*

Code	Description
400	Signature not valid. Bad request.
403	Forbidden call in cause of missing or invalid client certificate.
406	Data format or content is not valid.
409	Data already exist.
413	Payload to large.
500	Are not able to write data. Retry please.

Figure 21: Upload Responses

ℹ The batchTag in the response is the same as in the request, which is helpful to support parallel requests.

ℹ The 207 Response contains a document which tells the receiver more about successful or unsuccessful operations. In this document, the API returns the index of the key within the batch.

5.4.4 Transmission Protocol

During the upload, the uploader identity is extracted from the client certificate. If the client certificate is valid, the submitted content is validated, split and stored in the database. The size of the payload is limited to avoid to big requests.
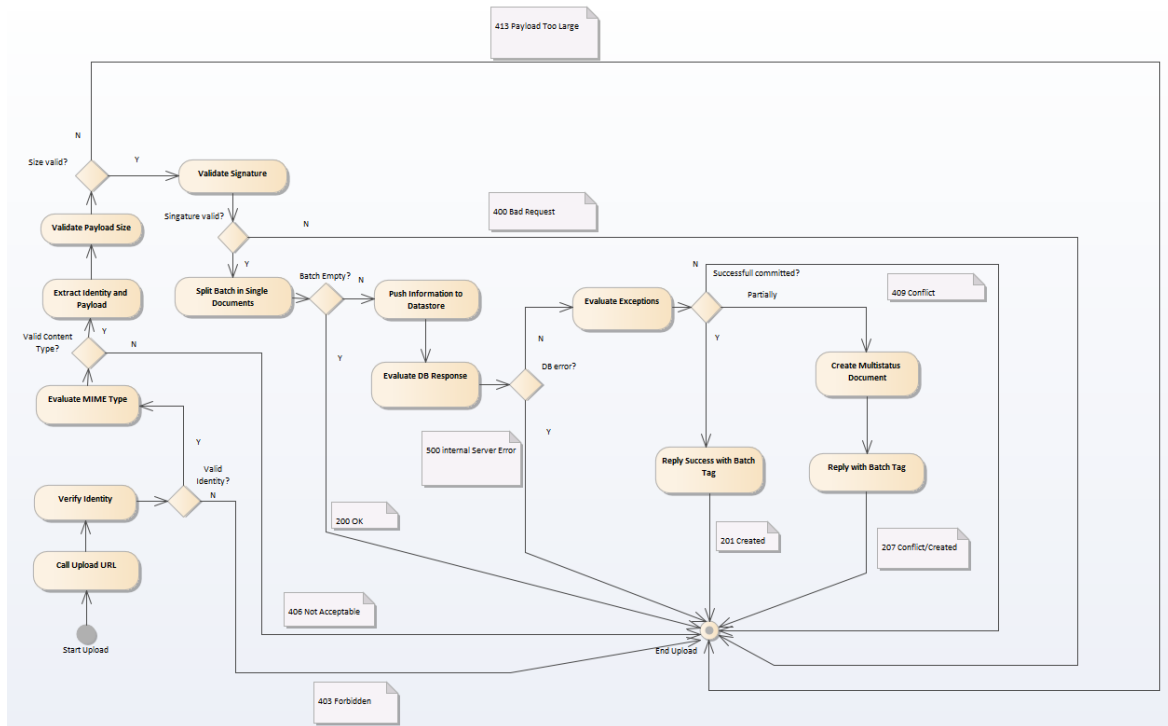
Figure 22: Upload Transmission Process

ℹ The API returns a "batchTag" to uniquely identify the uploaded set. This is necessary to support a release process of uploaded keys in future versions.
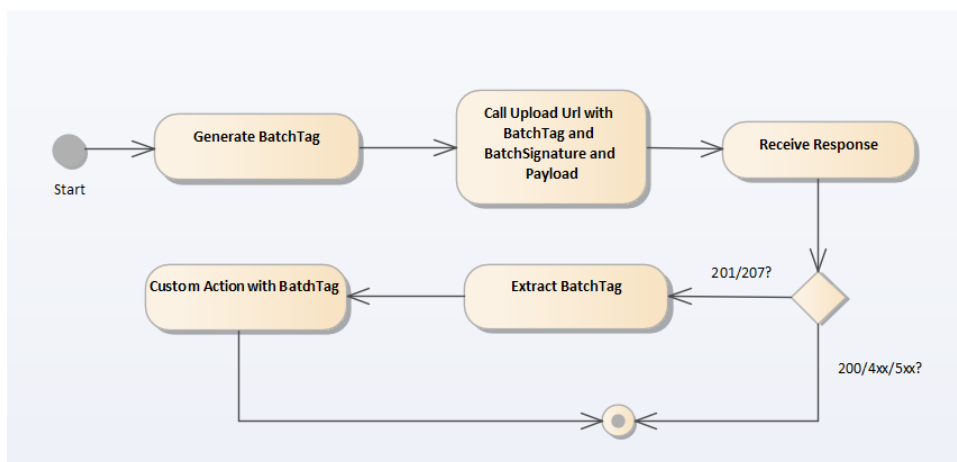
5.4.5 Client Process

Figure 23: Upload Client Process

5.5 Traffic Volume Estimates

5.5.1 Daily Incoming Traffic on Federation Gateway Service

We estimate the amount of data uploaded to the Federation Gateway Service during a 24-hour period, assuming a very bad pandemic situation and complete pan-European participation in this scheme. The basis of our estimate is the upload size of a single key including metadata, which is less than 200 bytes.

Each currently infected user uploads one key, while a newly infected user uploads up to 14 daily keys of the past two weeks. Hence, we need the current number of infections (say, 1M—the total cumulative number of reported infections in Europe and Russia, as of June 2020, is less than 2.5M) and the rate of daily new infections (say, $0.01\% = 10^{-4}$, which is large). Let's assume the European population at 750M and virtually complete app adoption. This gives $14 \cdot 10^{-4} \cdot 750 \cdot 10^6 = 1.05 \cdot 10^6$ new diagnosis keys and 1M other diagnosis keys per day, summing up to roughly 2.05M keys in total.

Consequently, the Federation Gateway Service receives $2.05 \cdot 10^6 \cdot 200\ bytes \approx 390\ MB$ per day, most of which has to be downloaded by each participating country.

ⓘ In theory, higher values are possible. This is a pragmatic upper bound; we expect much lower values in practice. Factoring app adoption rates below 75% and significantly lower infection rates than assumed above, daily volume won't exceed 100 MB. Moreover, the precise numbers vary somewhat, depending on formatting, frameworks, header compression, batch size, and other technical details.

5.5.2 Daily Traffic Between National Backends and Their Users

Not all keys need to be distributed to everyone. Depending on the size of a country, the rate of cross-border travel, the relative number of visitors from other countries, and epidemiological factors, the relation between domestic keys and foreign keys varies greatly.

5.6 Callback Interface

5.6.1 Overview

The callback interface consists of three operations for managing callback URLs:

GET `/diagnosiskeys/callback` Gets the current callback URLs.

PUT `/diagnosiskeys/callback/{id}` Put or Update new callback URL.

DELETE `/diagnosiskeys/callback/{id}` Delete callback URL.

Figure 24: Callback Interface

With this operation, it's possible for each national backend to register a callback GET operation which receives data changes—this way, there's minimal lag between new uploads and downloads. The Federation Gateway Service acts virtually as a forwarding gateway.

The API will append the parameters "batchTag" and "date" to the query; compare the example below.

Provided by backend:

> https://national.backend/notify_me

Called by the Federation Gateway Service:

> https://national.backend/notify_me?batchTag=dbg34924jfdnn&date=04-03-2020

The national backend is informed by the callback function that a new batch, tagged dbg34924jfdnn, is available since 04-03-2020. (And no, a more precise timestamp isn't necessary—for each day, any batch can be uniquely identified using the batchTag.)

⚠️ The Federation Gateway Service performs mutual authentication with the national backends. This means the API validates the provided server certificate of the national backend and provides its identity as a client certificate to them. Each national backend has to explicitly whitelist this identity and has to provide a server certificate public key to the Federation Gateway Service for whitelisting.

5.6.2 Parameters

The GET operation has no parameters.

The PUT operation contains the parameters for ID and URL:

Figure 25: Callback Put Parameters

Delete Operation:

Figure 26: Callback Delete Parameters

5.6.3 Responses

GET:

Figure 27: Get Response

PUT:

Figure 28: Put Response

DELETE:

Code	Description
200	OK.
403	Forbidden call in cause of missing or invalid client certificate.
500	Are not able to write data. Retry please.

Figure 29: Delete Response

5.6.4 Transmission Protocol

Registration   Flow:
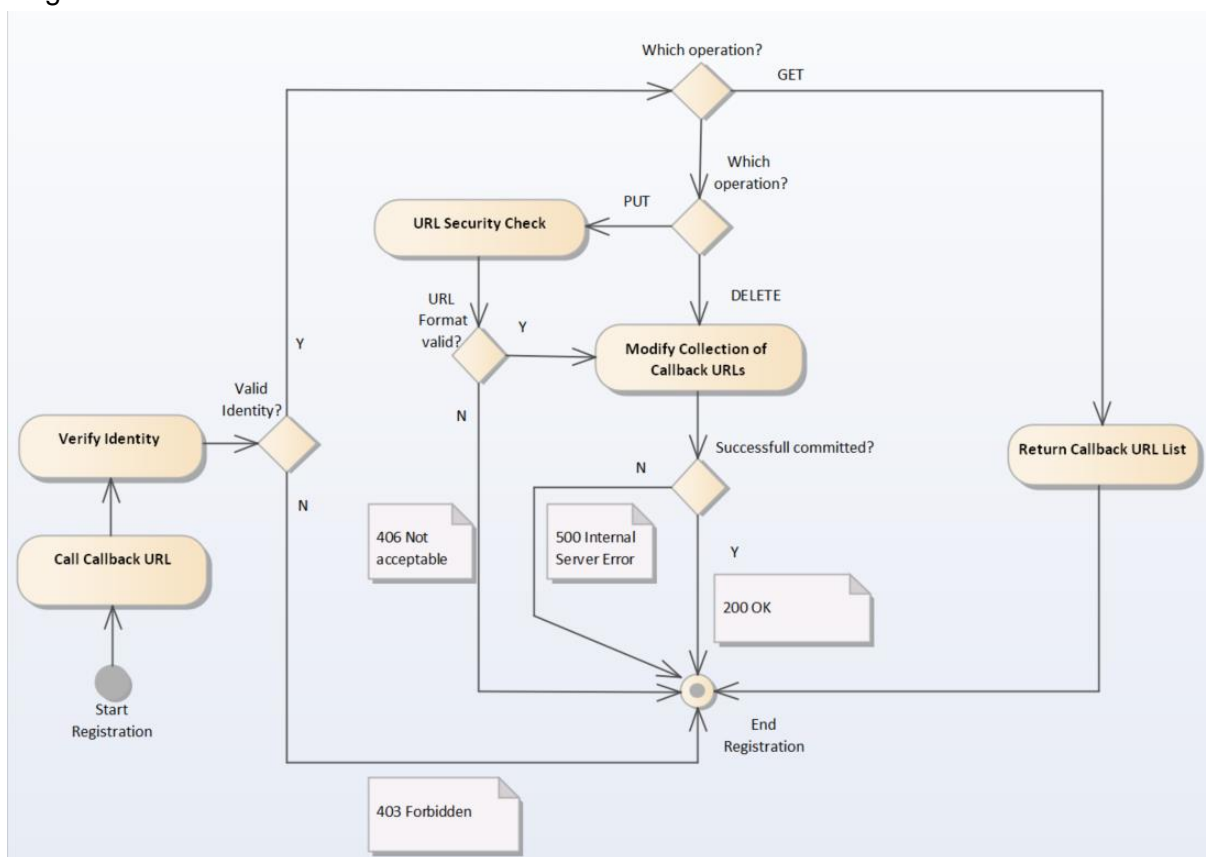
Figure 30: Callback Registration Flow

Callback Flow:

If a new batch of diagnosis keys was received, the API calls all registered URLs to signal that there is a change for a special batch and date.
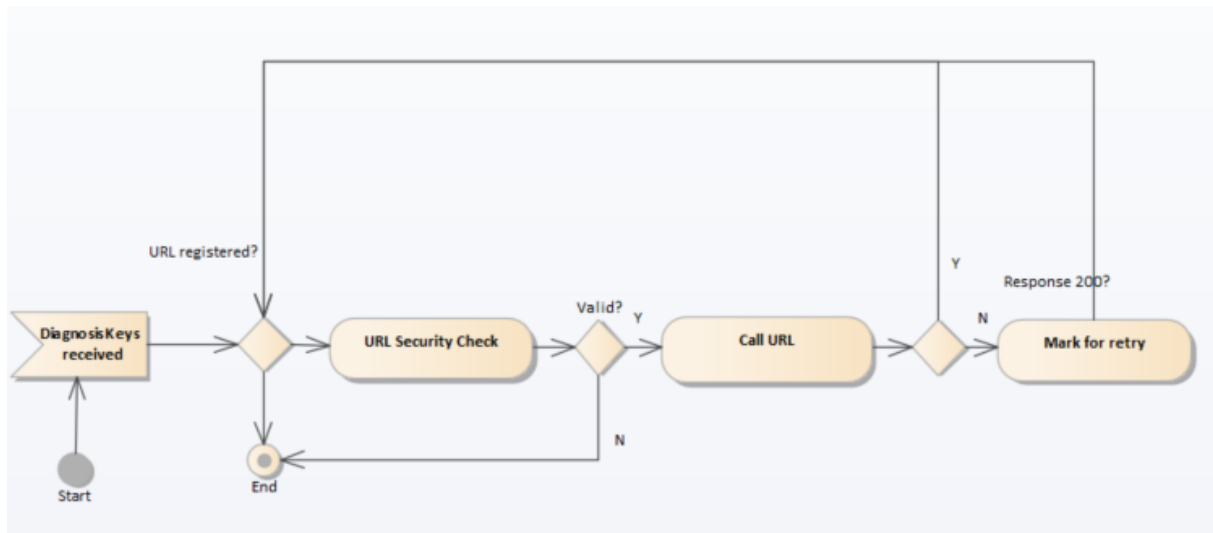
Figure 31: Callback Flow

ℹ️   The API remembers the last downloaded batch of a backend. If a backend downloads a later batch the Callback URL is not executed.

5.6.5   Client Process

On clients side the callback URL is called with batchTag and date. The national backend can execute then custom logic or download the data directly.
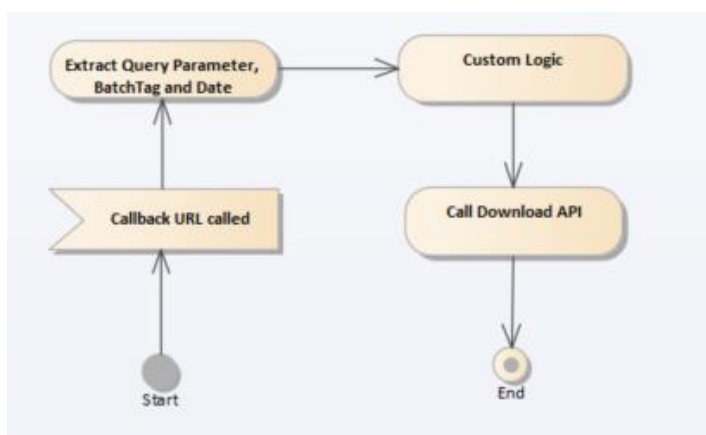
Figure 32: Callback Client Flow

5.6.6 Security Considerations

The callback interface needs to verify the given URLs during the registration and before the execution. Mandatory checks are:

Check	Reason
HTTPS	Non-HTTPS connections are rejected. No FTP, gopher etc.
Local Addresses	To avoid the execution of internal services, the given address must be checked for non-public addresses.
DNS Checkup	The resolution of the HTTPS addresses needs to be checked for non-public addresses.

Table 5: Callback Security Checklist

5.7 Audit Interface

5.7.1 Overview

The audit interface contains operations to audit parts of the service by the users from outside to validate the integrity of the running system.
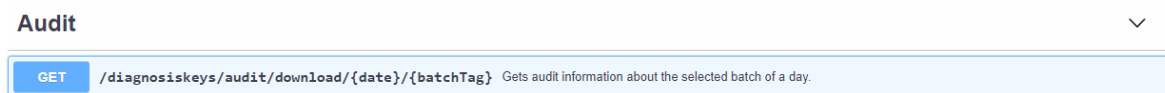
Figure 33: Audit Interface

5.7.2 Download Audit

This audit operation provides the possibility to verify data integrity within a batch. The operation returns information about the batch, for instance:

- Countries contained in the batch
- Batch signatures by country
- Uploading Information

All this information can be cross-checked over the certificate authority.

5.7.2.1   Parameters

Name	Description
date * required string (path)	Date of the download. 2020-05-01
batchTag * required string (path)	batchTag of the selected date. 123456
Accept * required string (header)	application/protobuf+v1.0

Figure 34: Download Audit Parameters

5.7.2.2   Responses

| Code	Description

200	OK. Returns the audit information to the selected batch.
	
	Media type
	application/json+v1.0 ▾
	Controls Accept header.
	
	Example Value
	
	``
	{
	  "NL": {
	    "batchSignature": "...",
	    "uploadedTime": "...",
	    "uploaderThumbprint": "..."
	  },
	  "IT": {
	    "batchSignature": "...",
	    "uploadedTime": "...",
	    "uploaderThumbprint": "..."
	  }
	}
	``

400	Invalid BatchTag used.
403	Forbidden call in cause of missing or invalid client certificate.
406	Data format or content is not valid.
410	Date for download expired. Date does not more exists.

Figure 35: Download Audit Responses

⚠️ The batchSignature in the response is calculated over all keys within the batch by country of origin. For verification purposes, it is necessary to hash all keys of each country and check this against the signature of the same country. Hashing all keys across different countries won't work.

6 Security

Security consists of three major components, commonly dubbed *confidentiality*, *integrity*, and *availability*. A single Federation Gateway Service, if designed correctly, covers all these components perfectly.

6.1 Confidentiality

Confidentiality refers to the requirement that only approved users—in this case, the national backends—can access the service, and that the service can be identified. This is achieved by using both client certificates (to authenticate clients) and a server certificate (to authenticate the server where the service is running). Additionally, client backends will be subject to certificate whitelisting, that is, only accepted client certificates can access the server.

Hence, authentication is mutual. The server proves his identity to the client and the client provides a client certificate to the server. Both can verify their identity via a certificate authority (CA). After this authentication process, the identity is retained for the duration of the session.
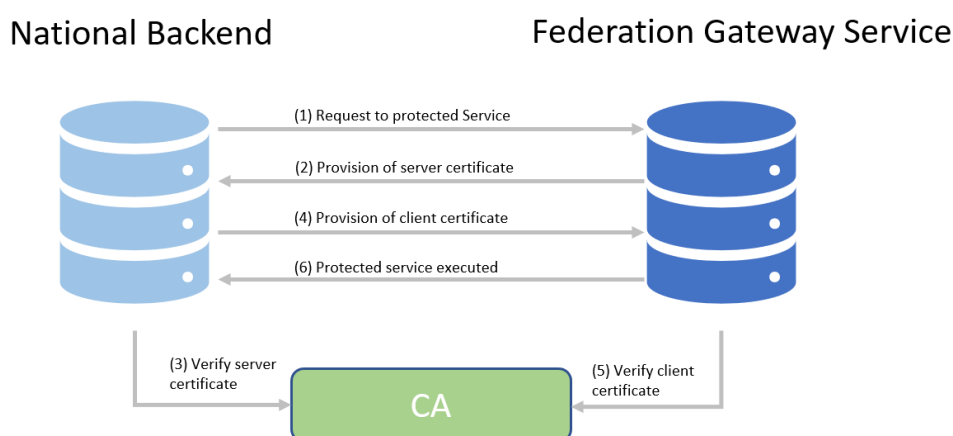
Figure 36: Backend Confidentiality

6.1.1 Certification Process

The provider of the Federation Gateway Service orders a server certificate from a certificate authority and the clients order client certificates from a certificate authority as well. Both parties exchange and integrate the certificate information manually by whitelisting in the systems.

6.1.2 Certification Authority

As certification authority (CA), Verizon can be used to order the certificates by each country/app. It's also possible to use other CAs.

i To improve security, we favor a dedicated CA which alone provides all certificates to the Federation Gateway Service and the national backends, because these certificates are then wholly controlled by this dedicated CA and will only be issued to accredited participants.

6.2 Integrity

Integrity refers to the requirement that data structures and content—either accidentally or maliciously—won't be compromised. This is achieved simply by verifying client identity and checking the uploaded data for validity. Since the data stored by the roaming service is kept locally encrypted and read-only, validity of the downloaded data is guaranteed.

i It improves trust and integrity, if each national backends signs each key. Thus, uploaded data can be validated by each downloader. Note that this step increases the traffic and validation overhead.

6.3 Availability

Availability refers to the twin requirement that the service delivers a guaranteed uptime (as a percentage of time) and a guaranteed performance (as a maximum response time). Both these demands can be met by running the service on any of the state-of-the-art cloud environments, which provide elastic compute power, sufficient storage and bandwidth, and all the necessary defense mechanisms against malicious attacks, natural disasters, and the occasional accident.

Moreover, we suggest deploying the Federation Gateway Service in at least two geographically separate zones.

7   Technology Choice

Component	Technology	Core Features
Container Platform	OpenShift	Kubernetes-based: high availability and scalability
REST API	Express (Node.js)	Powerful, lightweight web Framework
Distributed NoSQL Database	MongoDB	TTL index, Object ID, secondary indices, aggregation pipelines, changed streams
Load Balancer	HAProxy	Reverse proxy, load balancing, detailed traffic metrics, SSL offloading
Web Server	Nginx	Reverse proxy, lightweight web server

Table 6: Technology Proposals
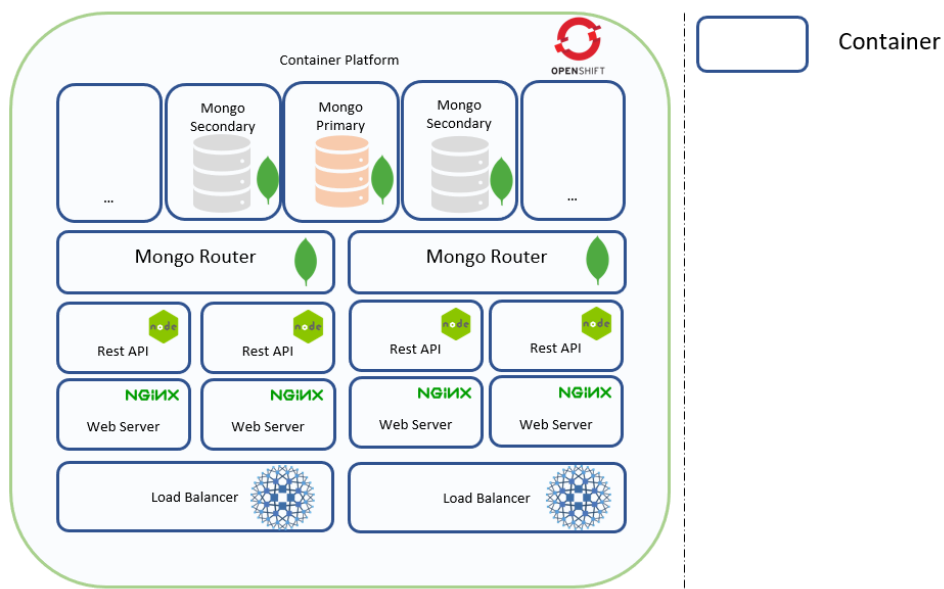
Deployment Example:

Figure 37: Deployment Example

8   Auditing

8.1   Overall

To ensure the auditing conditions, all requests to the Federation Gateway Service passes an audit module creating an audit log, which produces log files, event streams, or tables within the database. This data can be displayed on a dashboard via standard visualization tools like Tableau, Kibana, Splunk, Grafana, etc.

8.2   Data Privacy

Data privacy is being heavily discussed in all EU member states. There are lots of different laws and concerns about medical data sharing, overshadowed by the GDPR. The auditing mechanism should reflect these concerns from a technical perspective to ensure:

1) Data processing in compliance with GDPR
2) Risk minimization of unauthorized access
3) Protection of the rights of the data subject

This can happen in several ways:

- Client certificates to verify the identity of the national backends
- An active trust mechanism—backends may choose whom to trust (whitelisting) or not to trust (blacklisting)
- Logging of data access
- Encryption in transit using TLS
- Encryption at rest in the database
- Intrusion detection and abuse alerts

i   Has to be specified more detailed after EDPB has published its opinion on document version 0.9.

8.3   Data Transmission

Client information is extracted from the client certificate, the requested or submitted data (key origins, key destinations), and the timestamp of the operation. This information is used to create statistics about the clients and the traffic details. Consequently, all uploaded and downloaded information is guaranteed provable.

8.4   Traffic

Traffic can be monitored in three ways:

1. Via database logs

2. Via database functionality (manually)
3. Via external event-based monitoring tools

9 DP3T Compatibility

The Federation Gateway Service is fully compatible to DP3T's publishing/feed system, provided the following conditions:

- The formats used by DP3T have to be converted to the Federation Gateway Service data format by the DP3T publishers
- The Federation Gateway Service doesn't actively pull the DP3T feeds

From another perspective it's also possible to implement the used data formats directly in the Federation Gateway Service, e.g., with a new input format type:

application/dp3t+v1.0

Concerns regarding security, Bluetooth communication, and other details are not part of this architecture consideration.

More information about the specification can be found here:

https://github.com/DP-3T/documents/raw/master/DP3T%20-%20Interoperability%20Decentralized%20Proximity%20Tracing%20Specification%20(Preview).pdf

10 Alternative Data Exchange Methods

If a single European Federation Gateway Service, run in a suitable cloud environment, cannot be agreed upon for political reasons, the Federation Gateway Service can also be implemented in a distributed fashion using either of two different technologies: mirroring or a blockchain.

Both technologies offer neither better performance nor more security, and they're both adding an additional layer of complexity. Nevertheless, a storage solution that is distributed across several or all participating countries may be the preferred solution for some policymakers.

10.1 Mirroring

Mirroring lifts the idea of database synchronization for load balancing and disaster resistance to a higher level. Instead of the built-in capabilities of a single cloud environment—which includes load balancing and replicated databases (see section on database structure)—a list of available mirrors in different countries is the starting point. Each session between a national backend and one of the mirrored Federation Gateway Services needs to be replicated across all other servers, which leads to small inconsistencies, especially if download intervals are large.

For instance, imagine the backend of country A uploads a batch of new diagnosis keys to the Federation Gateway Service in country A just seconds after the backend of country B downloads the new diagnosis keys from the Federation Gateway Service in country B. Now, depending on the interval until the backend of country B downloads fresh data, it can't see the new data from country A's backend, potentially for hours.

Granted, this lag could be reduced by sending notifications about new uploads across the different Federation Gateway Services, but this once more complicates the architecture, increases the amount of traffic, and introduces new pitfalls that aren't there in the case of a single European Federation Gateway Service.

10.2 Blockchain

A blockchain has the twin advantage of providing hamper-proof distributed data storage and catering to the yearning after new technology. Since data lifetime is restricted to 14 days, the usual downside of any blockchain—scalability—is not an issue. Nevertheless, blockchain technology arouses as much doubt and criticism in some as it produces enthusiasm in others. Just like mirroring, it introduces needless complexity and synchronization lags.

APPENDIX

(A) Authentication:
- Connections to the Federation Gateway Service (FGS) will be over HTTP using TLS with cryptographic settings that meet or exceed the relevant ENISA recommendations on algorithms, key sizes, and parameters.
- There will be mutually authenticated TLS connections between the FGS and each national backend.
- Trust validation happens by means of certificate validation based on TLS client, TLS server certificate, and server name (CN/subjectOtherName, according the CAB forum standard).
- This is combined with pinning by both parties on explicit certificate, a dedicated CA in the chain or issuing CA by the FGS.
- For this reason, each national backend will inform the FGS of the Certificate in the chain below which they consider any client certificate as being authorized by the national backend to connect to the FGS on their behalf.
- In the most extreme case, this may be just the actual leaf certificate or a self-signed certificate. In general implementers are urged to provide a (dedicated) CA certificate as to minimize operational logistics (from the perspective of the gateway operator) around key rollover, revocation and general long term certificate management.
- The operator of the FGS will communicate the certificate in the chain below which they consider any server certificate as being appropriate for the FGS.

(B) Digital Signature TEKs:
- Contents submitted to the FGS will be digitally signed by the national backend.
- For this reason, each national backend will inform the FGS of the certificate in the chain below which they consider any client certificate as being authorized by the national backend to sign their domestic TEKs on their behalf.

(C) Operational and Runtime Considerations:
- The national backends will communicate a contact point for operational matters if such is not readily evident from the certificate.
- The operator of the FGS will communicate the certificates used by each national backend to all other national backends.

REFERENCES

[1] Google and Apple: Exposure Notifications – Roaming Approaches (May 5, 2020).pdf

[2] CWA Project: "European Interoperability Conceptual View", Version 0.02 from of 22th May 2020.

[3] Ulrich Luckas, et al.: „Interoperability of decentralized proximity tracing systems across regions" – version 2.1 (7 May 2020)

REFERENCES