Proposal: DP-3T Backend Specification

Version: 0.1 / 2020-04-29

Authors:

Dirk-Willem van Gulik
Tim Brouwer
Ryan Barrett

Introduction

This proposal is based on the D3PT API specification found here.

The aim of this proposal is threefold: firstly to improve the existing specification by including more details, such as extra HTTP headers. This helps to ensure compatibility between implementations (For design 1 & 2).

Secondly, we propose an API to allow for a more efficient distribution of the Cuckoo Filter binary data (Design 2). These filters will be downloaded by potentially hundreds of millions of users daily so there is a real benefit to doing so efficiently.

And finally we want to document some non-functional requirements expected of a likely system; including the need for ‘static’, bulk CDN based distribution of certain data.

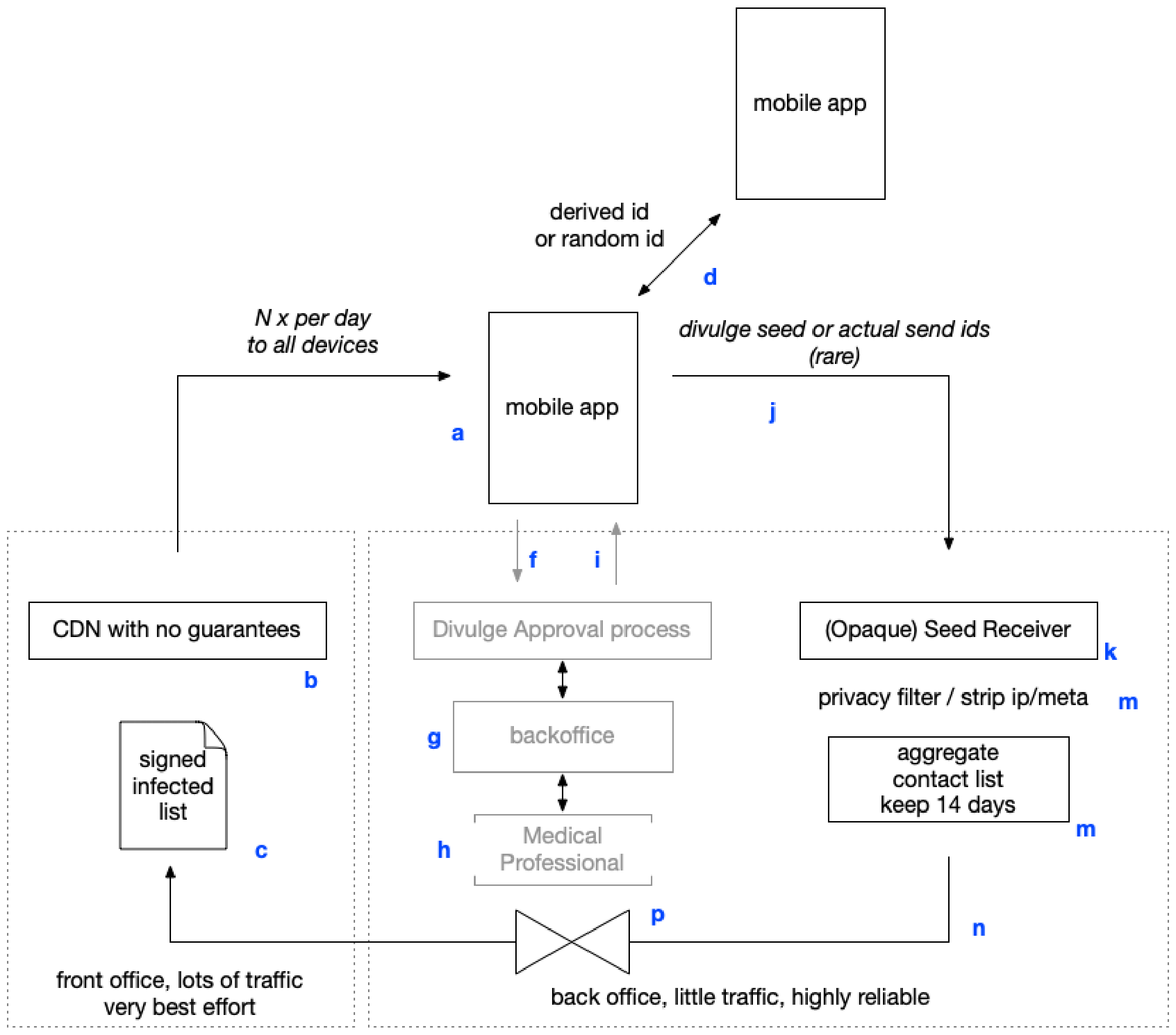
To achieve this we describe several variants of the implementation that can be used. The specific environment of the solution should be taken into account to decide on a specific implementation.

Operational setting

Logical/abstract architecture - to help discuss the non-functional, information security and compliance aspects of any implementation. And help guide what needs to be arranged in terms of governance.

There are three systems/backends in play.

- A CDN like bulk publishing system (b, c) that pushes out the infection lists to all mobiles participating, every day.
- A system that collects the data from just the (infected) mobiles, just once per infection
- [optionally] An approval/filtering process that ensures that only validated infections make it into the official list for those countries that require, say, a test or a professional assessment.


mobile app
derived id or random id
d
N x per day to all devices
mobile app
a
divulge seed or actual send ids (rare)
j
f
i
CDN with no guarantees
b
Divulge Approval process
(Opaque) Seed Receiver
k
privacy filter / strip ip/meta
m
g
backoffice
aggregate contact list keep 14 days
m
signed infected list
c
h
Medical Professional
p
n
front office, lots of traffic very best effort
back office, little traffic, highly reliable

A Mobile app(**a**) installed millions of devices does regular (e.g. 1-4 times per day) fetches through a common Content Delivery Network (CDN), with no particular integrity or reliability guarantees (best effort)[^1].

Fetched is a (possibly digitally signed file (**c**)) with the contamination details of the day. This file is generated by a very reliable central backend (**m**) and contains just the relevant information (**p**).

At some point (e.g. on a national scale, several 10's to 100's of time an hour) a medical professional will ask if his patient is willing to divulge his contact (process **f,mg,h,i**) or something or someone decides that an infection needs to be reported.

The device then posts, **j**, plain http(s)/REST, its opaque contact data to the receiver(**k**). Depending on the protocol - this is either a seed or the actually broadcasted IDs (**d**).

This is filtered, at pure IP level, by (**k**); where no data is kept but a short log and a few percent of the net-flows.

The list of the last 14 days is then filtered, if appropriate, before it is delivered every few hours or every day to the distribution area.

Ramifications for the API

This means there are three areas with very different properties.

- CDN: bulk, only handles public data, best effort, needs to be hit by the whole population. very few privacy concerns beyond the usual ones of seeing IP's and headers. Should respond fast/efficient. No state, no need for logging or backups.
- Receiver: little traffic, but should never ‘lose' data and be much more privacy sensitive - as there is an implied; this person may be infected. Batch oriented; so no speed or other concerns; very clear data retention expectations
- Is Infected Approval process (optional): process that supports medical processes - and may be highly sensitive - but largely based on existing processes, medical records, Laboratory Information Management Systems (LIMS) and so on; and interface is just an 'ok' to publish.

This allows for API optimization. So it is recommended that there are at least two URI prefixes; a publication URL and a submission URL.

Proposed endpoints

Based on progressing epidemiological insight, health authorities may decide to change the rate at which keys are distributed, prioritise certain sets of keys or how keys are bundled together.

To enable such a change by only reconfiguring the server side, we proposed a level of indirection for the main bulk distribution endpoint. Where first a list of URLs & last updated dates is fetched; and then in a second fetch (using HTTP pipelining where possible) the data needed.

This level of indirection gives authorities options to prioritize some batches over others, and to some extent influence risk calculation done on the client device.

We propose supplying this 1) by `date since updated’ and 2) that it is the health authorities responsibility to limit the sets published at any given time to the ‘max’ they want to be applied ‘as soon as possible’ to the known contact lists of that day.

This can be used to optimize distribution in times of high load. For example in the following cases, which can become relevant when the epidemic flares up again:

- Health officials might provide updates in smaller time intervals, which all need to be delivered to the clients as quickly as possible.
- Health officials might decide to (de)prioritize data from certain regions *or population cohorts*, on epidemiological grounds.
- Reduced or intermittent network throughput due to technical problems.
- They may change the number of days covered by the data published without requiring changes in the field.

Be aware that in the last case this implementation might have an effect on the deanonymization risk of the DP-3T implementation as a whole.

Cross border Considerations

Our society and public health services are organised around geographical borders. Unfortunately a virus such as COVID-19 does not respect borders.

See Appendix Federation for an pan-European extension of above principle.

Endpoints -- CDN / bulk distribution

Design 1: simply download the list of seeds.

```
    for each (entry in seed list)
        Generate the list of EphIDs for the correct range
        for each EphID generated
            Check if this matches an entry in your local store
```

Design 2: To check for infection: first load the list of cuckoo-filters from /exposed.

```
for each (cuckoo-filter in cuckoo-filters)
    load data from `uri_filter` into your cuckoo-filter
    run your local interaction emphIDs through the cuckoo-filter.
```

Optionally - if the design its false-positive is set too low

```
    if ( any emphID are found in the cuckoo-filter )
        false positives are possible, so download (the relevant part of)
the full list of ids from uri_data and check them
```

Bulk Endpoints

Please refer to the D3-PT API specifications for details on the parameters and message content.

Below end points are all for the CDN / bulk distribution.

We suggest to move the endpoint for a device to -publish- a seed to a separate URL and interface. So, if so desired, it can be decoupled and routed to a different operational setup (see appendix C).

Non functional expectations are

- Handle at least one fetch/day from the mobiles (millions of requests per day, thousands per second).
- Very resistant to repeated/superfluent fetches
- Bad behaviour of one should not jeopardize *delivery* to others.
- Little or no security
- Tamperproofing is desirable - but for most protocols not uber critical as the match-space is very large and thus resistant against false positives. The

tampering is a risk for not reporting infected people that they are infected (which is in fact what some device owners may well desire, e.g. to keep their jobs).

general option/concern

We suggest to sign all responses with a RFC 3161 timestamp/signature. These can be prefixed safely as the DER block starts with an identifier/length specification. In that case - we suggest changing the content type to ‘application/covid-signed’.

Version 1, Design 1, Design 2

There are two designs; Design 1 and Design 2. All URIs are prefixed by /v1/ for version 1.0 and /d1 or /d2 for the second design.

BULK Endpoints summary

1. GET /v1/d1/exposed -- returns a list of URLs with JSON seed files and pub-Dates
2. GET /v1/d2/exposed -- returns a list of URLs with serialized CF filters and pub-Dates

In general - the URL lists returned by the above 2 calls may point to any URL; but suggested are to use a URL with the date of publication -or- the SHA256 of the file.

3. GET /v1/d1/exposed/<dateOrSHA256> -- returns a JSON array of seed/onset dates
4. GET /v1/d2/exposed/<dateOrSHA256> -- returns a serialized CF filter

It is suggested that the publication of the seeds is not handled by the bulk endpoint:

5. POST /v1/d1/exposed -- dropped (see Appendix C)
6. POST /v1/d2/exposed -- dropped (see Appendix C)

POST /v1/d1/exposed, POST /v1/d2/exposed

We propose to drop this endpoint from the CDN/bulk endpoint - see appendix C for a discussion.

GET /v1/d1/exposed, GET /v1/d2/exposed

Content-Type

```
Content-Type: application/json; charset=UTF-8
```

ETag

```
ETag: "hash or similarly unique identifier"
```

Server assumed to support ‘get-if-modified’ and similar cache optimisation.

Message content

```
{[
      {
            date: '2020-04-20T19:30:16Z',
            uri: '/exposed/data/2020041634'
      }, ...]
}
```

Returns an json array containing a list of data batches in reverse-chronological order - so the latest update comes first. Each entry consists of a date as a string; that should be encoded as a ISO 8601 gregorian date (rooted to the 20 May 1875 for leap-day/second purposes) following the YYYY-MM-DD format[1] in UTC. And a URI that is again a string and should contain the octet, wire-order encoded URL of the data.

It will never contain more entries than needed for a `empty’ client to fully catch up[2].

Any dates in **this** response related to the date of publication. So an application can simply fetch any URI which has a date that is newer than the last it has fetched; or all of them if it has never fetched any. This response is guaranteed to never return more URLs than needed by the client in any circumstance. The date in this response is purely the publication date - it say nothing about the date of the data the data refers to.

The URI is opaque and may be both a full RFC2396 style full URI or a relative path URI. In this case the path is relative to the path of the get request used.

[1] We’re following the definition of the DP3T whitepaper -- it may be prudent for applications to anticipate a full date & time string; down to seconds with a timezone.
[2] I.e. a client can alway safely fetch all; always process these files immediately and then discard them. And then, from then on, only fetch newer files (according to the date from this response). It thus does not need to have a reliable time or use the HTTP Date header.

Applications should anticipate the use of opaque URL (e.g. a UUID or a SHA256 as its name) and expect (and honour) temporary or permanent HTTP redirects.

The array contents is extensible should there need arise to add more metadata.

Design note on the dates / API contract:

Note: there are two dates; the date on which the data was made available / a fetch was made; and the date(range) associated with the data itself. As it may be the case that authorities need to temper the flow of data; or prioritise certain data (e.g. of a region of a specific time slot) - it may well be that data published on day N contains data about infections on several days (this is generally true as tests take time).

So the dates alluded to in A & B are about the date of publishing-fetching.

Or in other words - the API contract is simplified. The application tries to be as up to date relative to the data published. And once published strives to most efficiently fetch it to stay up to date. It is then to process this data as quickly as it can (and then discard it as quickly as it can).

And the health authorities rule is simplified to publishing what is most prudent to publish at any given time; and never publish more than is needed.

GET /v1/d1/exposed/<dayDateStr>

Content-Type

```
Content-Type: application/json; charset=UTF-8
```

Accept-Ranges

```
Accept-Ranges: bytes
```

Allows requests for partial ranges, allowing for resume-support.

ETag

```
ETag: "hash"
```

The hash identifies the binary data and is used for change-checking. By including a strong ETag we allow for the CDN/cache to work efficiently and support partial downloads.

The hash itself is the same sha-256 as described in the specifications.

Messages/Payload

V1/D1

A JSON array that contains a list of Secret Keys; each entry consists of two fields; the ‘key’ and the ‘onset’. With the KEY a base64 encoded string of the 32 secret bytes. And the “onset” a ASCII encoded string of ISO 8601 gregorian date (rooted to the 20 May 1875 for leap-day/second purposes) following the YYYY-MM-DD format[3] in UTC.

V1/D2 CuckooFilterFormat - v1.00

[3] We’re following the definition of the DP3T whitepaper -- it may be prudent for applications to anticipate a full date & time string; down to seconds with a timezone.

A binary format; a 32 byte header followed by buckets of slots of a specified bit length. The server optimises these buckets, slots and bit lengths (for low, or neglectable, false positives and size).

The key shall be the 256 bit / 32 byte hash as resulting from 2020/4/12 version of the Whitepaper: H(TRUNCATE128(H(seed))||i). No further hashing is required.

The key has 32 bytes which are used as follows:

```
byte  0..3         up to <bits-hash> bits for the LSB of the Cuckoo hash
byte  4..7         up to <bits-hash> bits for the MSB of the Cuckoo hash
byte  8..31        up to <bits-verify> bits.
```

Where `<bits-hash>` and `<bits-verify>` are set as low as is feasible given the acceptable false positive rates by the server.

The Cuckoo filter shall be serialised as a header followed by the table. The header is 4x4=16 bytes in size.

```
Magic string 4     bytes D3, D3, 3D, 3D
Version            1      byte, top 4 bits major, bottom 4 bits minor.
                          Currently set to 0x10 for version 1.0
<Depth>            1      byte, fixed to 4
<bits-hash>        1      byte, without the occupied bit.
<bits-verify>1     byte
<N buckets>        4      bytes, unsigned 32 bit integer, network order
<N slots>          4      bytes, unsigned 32 bit integer, network order
```

This 16 byte header is followed by

```
<N buckets> with each
      <Depth> slots with each
            sequence of ( bit-hash + 1 )/ 8 rounded up bytes:
                  is occupied       1                 bit
                  partial hash      bits-hash         bits
                  padding           *
                  any padding bits set to 0 if bits-hash + 1 not a multiple of 8
            verify hash             bits-verify  bits
                  padding           *
                  any padding bits set to 0 if bits-verify not a multiple of 8
```

With the partial hash being limited to the number of bits needed for N buckets.

So this, 1.0, version of the serialisation does not pack the bits; both hashes are padded to a full byte. Version numbers are semantically meaningful.

Appendix Cross Interoperability

Our society and public health services are organised around geographical borders. Unfortunately a virus such as COVID-19 does not respect borders.

To allow for the contract tracing to operate across borders a certain level of cooperation and standardization must be applied with regards to the solutions used, and those solutions must provide interfaces to make this cooperation possible.

Health services and their operational borders are usually, but not exclusively split by country. For the purposes of this discussion we refer to countries, however this can apply to any arbitrary split. The API will serve states, counties or cities as well as they serve countries.

The assumptions are that:

- The users will have one app installed, that from their home country. (The issue here is that it is logistically hard for a national app to appear in foreign app-stores; and it is hard for a national of one country to fetch an app from an AppStore that person is visiting[4])).
- If infected, the user should report themselves as infected one once.
- If infected, the user should report themselves as such in their home country.
- Users should be notified if they have come into contact with someone in countries that they have visited.

These requirements can be met in two ways:

- Merging the infection reports from other countries into the data provided on the Home Country feed.
- Include the infection reports from other countries into a feed provided by the Home Country with API to allow the applications to choose which countries filters they apply.
- Each country publishes the infection data to a feed for their own country. An API is provided in the D3PT specification to allow the client to find

Merging infection data is inflexible and will lead to a lot of unnecessary overhead for the majority of users. Adding separate data files from other countries into a feed whilst providing a method to filter removes that overhead. It, however, requires countries to make efforts to explicitly share data.

By far the most flexible and effective solution is to have each country publish their infection data to their own feed and only their own feed. By including a list {country, feed url }'s it becomes easy for the user to check their interactions against the infection data from any country. **More importantly, this is the option described in the white paper (version 12th April 2020, p12).**

[4] It usually requires faking an address/valid-zipcode and sometimes playing around with the credit card addresses.

With this in mind we propose the following API to support distribution of the infection authority endpoints:

/meta/authorities

Returns a list containing the infection authority per region/country. Where this api can be found so you can check your risks if you've been abroad.

```
GET /meta/authorities
```

```
application/json
```

```
{
        "NL": "https://….",
        "DE": "",
        ...
}
```

This API allows for authorities to be added, modified or removed dynamically at the cost of this falling under the responsibility of every individual authority to maintain the lists.

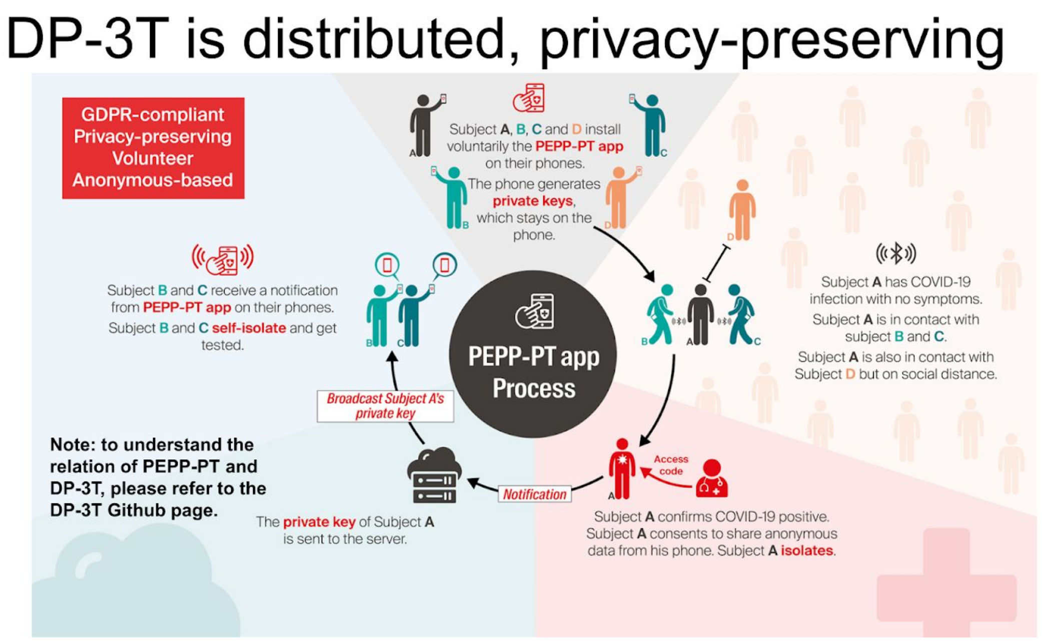
This API could also be exposed on a single URL, for example one under the control of an hypothetical global authority such as the DP3T project organization or the European Centre for Disease Prevention and Control.

Rather than using ISO 3167 country code - a very useful specifier may be the european NUTS[5] designator (level 1) which may to major socio-economic regions that often align with things such as work/home travel, etc. Also as ISO 3167 country codes often do not refer to geographically logical borders (e.g. the Netherlands has a border with Venezuela following this scheme).

[5] https://en.wikipedia.org/wiki/First-level_NUTS_of_the_European_Union,. https://ec.europa.eu/eurostat/web/nuts/background

Appendix Overview for app developers

Big picture overview for app developers who haven't read anything else


DP-3T is distributed, privacy-preserving
GDPR-compliant
Privacy-preserving
Volunteer
Anonymous-based
Subject A, B, C and D install voluntarily the PEPP-PT app on their phones.
The phone generates private keys, which stays on the phone.
Subject A has COVID-19 infection with no symptoms.
Subject A is in contact with subject B and C.
Subject A is also in contact with Subject D but on social distance.
PEPP-PT app Process
Access code
Notification
Subject A confirms COVID-19 positive. Subject A consents to share anonymous data from his phone. Subject A isolates.
The private key of Subject A is sent to the server.
Broadcast Subject A's private key
Subject B and C receive a notification from PEPP-PT app on their phones.
Subject B and C self-isolate and get tested.
Note: to understand the relation of PEPP-PT and DP-3T, please refer to the DP-3T Github page.
A
B
C
D

See https://github.com/DP-3T/documents/blob/master/DP3T-Slideshow.pdf

Design 1 DP-3T and TCN:

```
for each (cuckoo-filter in cuckoo-filters)
      take the seed; reconstruct the N broadcasted ID
            for each id reconstructed
                  check them in the list of local ids received.
                        stop if you are in it.
```

Design 2:

```
for each (cuckoo-filter in cuckoo-filters)
      load data from `uri_filter` into your cuckoo-filter
```

```
            check each of your local ids for presence in the cuckoo-filter.
                  stop if you are in it.
```

Optionally - if Design 2 is to carry metadata (like TCN and PACT) or if the filter is very small/'too' efficient

```
for each (cuckoo-filter in cuckoo-filters)
      load data from `uri_filter` into your cuckoo-filter
            check each of your local ids for presence in the cuckoo-filter.
            if ( interaction in cuckoo-filter )
                  download a partial list of ids from uri_data
                        stop if you are in it.
```

PACT and TCN:

```
for each (cuckoo-filter in cuckoo-filters)
      check them in the list of local ids received.
            stop if you are in it.
```

Appendix C -- submission interface

In most (national/large regional) settings it will be technically required to have a CDN handling the brunt of the millions of devices doing their exposed data fetching every day. While the submission of the seed data of (confirmed) infected patients is relatively rare and far between.

Furthermore there are various security and non-functional concerns around these interfaces; such as related to (D)DoS, stuffing, smear and so on. It is therefore likely that this interface will need to see additional security applied (such HTTPS and X.509 certificate pinning or client-certificates).

Therefore combining it with the bulk CDN interface is not ideal; as it would impose those costly requirements on the bulk fetches.

We therefore strongly suggest that these are kept on separate URIs and note that (see also the multi-country setup) a submission will generally go to a single backend - i.e. that of the issuing authority.

POST /v1/exposed

Endpoint for publishing the SecretKey.

Request headers

Content-Type

`Content-Type: application/json`

Content-Length

`Content-Length: ...`

Request body

Success Response status line

Protocol version

`HTTP/1.1`

Status code

`200`

Status text

`Succes`

Success Response headers

Success Response body

Notes