DP3T API

Documentation

April 20, 2020

Contents

Technical Description

Introduction

This document outlines the backend as it is. The models and requests are automatically generated. Hence, they should reflect the current live situation. We try to provide examples and description to clarify the use of the fields and responses returned.

Deployment architecture

From an architecture perspective - this API has two very different functions:

Acceptance of seeds for inclusion (`post /v1/exposed`) which occurs relatively rarely; and may be subject to very tight security controls (especially if it is in ahead of a medical test completing[1]).

From a systems perspective - this endpoint should be very reliable, and, especially, not loose data.

And privacy wise - one need to be somewhat careful as you have a vantage point of the IP addresses and similar identifying information.

Publishing of confirmed seeds (`get /v1/exposed/{date}`) is a bulk affair; where efficiency and maximum reachability counts. But where always up is not that key (apps will have to retry anyway) and where best effort is good enough (e.g. the data being a few minutes or hours stale is not an issue).

Here privacy is much less of an issue. And security is not really an issue from a confidentially perspective.

For this reason - real world implementors should anticipate that the two calls will generally be deployed on separate systems.

And hence live on separate URLs; one of which may be anycast or similarly 'virtualised'.

Deployment architecture

For all but the smallest countries; assuming daily fetches by a few million phones will require the use of a Content Delivery Network or similar 'dumb' delivery architecture.

For this reason - implementors should:

1. Assume that the hierarchy under `get /v1/exposed/{date}` is a static file style hierarchy.
2. Which[2] will require the regeneration of all 14 files at least once per 24 hours.
3. Assume that clients will use 'Get-if-modified' and the Date: data; and thus be very careful with things like system wide, stable, e-Tags.

Verification of Data

To handle heavy workload, requests are routed via a content-delivery-network (CDN). This means that we need to provide proof that the data was not modified by the CDN. We propose a Elliptic-Curve Digital Signature Algorithm using the P256 elliptic curve with a SHA-256 hashing algorithm. The P256 elliptic curve has good native support for the Apple and Android platforms to verify signatures. The public key should be available on the discovery platform and is as well included and distributed with the applications for iOS and Android.

[1] Most EU countries have a "right to not know" the results of a test, so it can be expedient to already provide the SecretKey ahead of time - and only push it onto the distributed list once a test result is in (the fact that the mobile app needs to reset its seed is not an issue; it either did it for good reason; or if the test came round negative - the rest is not an issue either).

[2] unless a binary/pyramid scheme is added to this proposal

The ensure the possibility of signature verification, the signed endpoints return an object with a signature and a data field, of which the data field contains a base64 representation of the list. In the current implementation the representation is a json of a list of keys. To improve performance of possible large decodings, we plan to switch to protobuf or something similar, which should speed up the parsing.

Since we only want to ensure that the data we are processing was indeed the data sent from the specified backend, it is sufficient to generate the signature of the content which will be processed.

Too further improve operability, the algorithm used to generate the signature should as well be encoded within the json object, similiar to a JWK (Json web key).

Suggestion: RFC3161 digital timestamp and signature

Another option would be the use of a simple RFC 3161 digital timestamp and signature. As this ties in well with existing infrastructure and is easy to deploy[3].

Google/Apple Privacy-Preserving Contact Tracing Similarities

[3] Signing side: https://interop.redwax.eu/rs/timestamp/ is just a handful of lines in a webserver config; client side- iOS and Android have support build in.

Web Service

Introduction

A test implementation is hosted on: https://demo.dpppt.org.

This part of the documentation deals with the different API-Endpoints. Examples to fields are put into the models section to increase readability. Every request lists all possible status codes and the reason for the status code.

/v1/

```
get /v1/
```

hello

Responses

200 Success

Type
string

/v1/exposed

```
post /v1/exposed
```

Endpoint used to publish the SecretKey.

Request Headers

Field	Type	Description
User-Agent *	string	App Identifier (PackageName/BundleIdentifier) + App-Version + OS (Android/iOS) + OS-Version

Request Body

Field	Type	Description
*	ExposeeRequest	The ExposeeRequest contains the SecretKey from the guessed infection date, the infection date itself, and some authentication data to verify the test result

Responses

200 Success

Returns OK if successful

Type
string

400 Bad Request

Key is not base64 encoded

/v1/exposed/dayDateStr

```
get /v1/exposed/{dayDateStr}
```

Path Parameters

Field	Type	Description
dayDateStr *	string	The date for which we want to get the SecretKey.

Responses

200 Success

Returns ExposedOverview, which includes all secretkeys which were published on dayDateStr.

Type
ExposedOverview

400 Bad Request

If dayDateStr has the wrong format

Models

All Models, which are used by the Endpoints are described here. For every field we give examples, to give an overview of what the backend expects.

ExposedOverview

Field	Type	Description	Example
exposed	Exposee[]	A list of all SecretKeys	c.f.Exposeemodel

Exposee

Field	Type	Description	Example
key *	string	The SecretKey of a exposed as a base64 encoded string. The SecretKey consists of 32 bytes.	QUJDREVGR0hJSktMTU5PUFFSU1RVVldYWVpBQkNERUY=
onset *	string	The onset of an exposed.	2020-04-06

ExposeeAuthData

Field	Type	Description	Example
value	string	Authentication data used to verify the test result (base64 encoded)	TBD

ExposeeRequest

Field	Type	Description	Example
key *	string	The SecretKey used to generate EphID base64 encoded.	QUJDREVGR0hJSktMTU5PUFFSU1RVVldYWVpBQkNERUY=
onset *	string	The onset date of the secret key. Format: yyyy-MM-dd	2019-01-31
authData *	ExposeeAuthData	AuthenticationData provided by the health institutes to verify the test results	TBD