ETSI GS E4P 008 V1.1.1 (2021-05)

Europe for Privacy-Preserving Pandemic Protection (E4P); Back-End mechanisms for pandemic contact tracing systems

Reference

DGS/E4P-008

Keywords

covid, eHealth, emergency services, identity, mobility, pandemic, privacy, security, smartphone

Contents

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Europe for Privacy-Preserving Pandemic Protection (E4P).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

The COVID-19 pandemic has generated significant challenge for many countries and their citizens and showed that digital technologies could play an important role in addressing this pandemic and future pandemics. Various applications, services and systems for contact tracing (identification and notification of those who come in contact with a carrier) have been developed in different regions.

Despite the similar goal of automated detection of COVID-19 exposure as a complementary solution to manual tracing (interviews with people diagnosed with COVID-19 to track down their recent contacts), their functionality, technology, scale, required data and limitations are different and may not interoperate.

These systems are currently being deployed in different countries and many more are expected in the near future. In particular, mobile devices with their contact tracing applications can support public health authorities in controlling and containing the pandemic. In that purpose, E4P has been created to provide a technical answer to pandemic crisis not limited to COVID-19 by specifying interoperable contact tracing systems.

1 Scope

The present document specifies back-end mechanisms for Pandemic contact Tracing Systems, including:

- architecture;
- information flow;
- protocols for sharing proximity data of contacts;
- the requisite APIs (Application Programming Interfaces); and
- privacy, data and system security.

Sufficient technical detail will be included to also facilitate means of interoperability in a later Work Item (ETSI GS E4P 007 [3]. Each Digital Contact Tracing System is characterized by its degree of compatibility with the E4P requirements work item (ETSI GS E4P 003 [1]).

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference/.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI GS E4P 003 (V1.1.1): "Europe for Privacy-Preserving Pandemic Protection (E4P); High level requirements for pandemic contact tracing systems using mobile devices".

[2] ETSI GS E4P 006 (V1.1.1): "Europe for Privacy-Preserving Pandemic Protection (E4P); Device-Based Mechanisms for pandemic contact tracing systems".

[3] ETSI GS E4P 007 (V1.1.1): "Europe for Privacy-Preserving Pandemic Protection (E4P); Pandemic proximity tracing systems: Interoperability framework".

[4] IETF RFC 5246: "The Transport Layer Security (TLS) Protocol Version 1.2".

[5] IETF RFC 2616: "Hypertext Transfer Protocol -- HTTP/1.1".

[6] IETF RFC 3629: "UTF-8, a transformation format of ISO 10646".

[7] IETF RFC 5905: "Network Time Protocol Version 4: Protocol and Algorithms Specification".

[8] ZIP File Format Specification, Version: 6.3.9.

NOTE: Available at https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT.

[9] Annex D.1 of NIST FIPS186-4.

[10] FIPS PUB 180-4: "Secure Hash Standard (SHS)".

NOTE: Available at https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

[11] "Exposure Notification Bluetooth® Specification", v1.2 April 2020, Google Apple.

NOTE: Available at https://developers.google.com/android/exposure-notifications/exposure-key-file-format.

[12] PRIVATICS team, Inria, France, Fraunhofer AISEC, Germany: "ROBERT: ROBust and privacy-presERving proximity Tracing, version 1.1", May 31st, 2020.

NOTE: Available at https://github.com/ROBERT-proximity-tracing/documents and https://hal.inria.fr/hal-02611265/en/.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] Frequently Asked Questions about the Corona-Warn-App.

NOTE: Available at https://www.coronawarn.app/en/faq/#anonymous.

[i.2] Criteria for the Evaluation of Contact Tracing Apps.

NOTE: Available at https://github.com/corona-warn-app/cwa-documentation/blob/master/pruefsteine.md#unlinkability.

[i.3] PRIVATICS team, Inria, France, Fraunhofer AISEC, Germany: "ROBERT: ROBust and privacy-presERving proximity Tracing, version 1.1", May 31st, 2020. .

NOTE: Available at https://github.com/ROBERT-proximity-tracing/documents, https://hal.inria.fr/hal-02611265/en/.

[i.4] G. Kessibi, M. Cunche, A. Boutet, C. Castelluccia, C. Lauradoux, D. Le Metayer, V. Roca: "Analysis of Diagnosis Key distribution mechanism in contact tracing applications based on Google-Apple Exposure Notification (GAEN) framework (version 1.2)", September 2020. .

NOTE: Available at https://github.com/3rd-ways-for-EU-exposure-notification/project-DESIRE, https://hal.inria.fr/hal-02899412/en/.

[i.5] Swiss Confederation: "Replay Attacks", June 14th, 2020, section "Unmasking users by eavesdropping EphIDs".

[i.6] Tijmen Schep: "Corona Detective".

NOTE: Available at https://www.coronadetective.eu/.

[i.7] O. Seiskari: "BLE contact tracing sniffer PoC".

NOTE: Available at https://github.com/oseiskar/corona-sniffer.

[i.8] S. Vaudenay: "Centralized or Decentralized? The Contact Tracing Dilemma".

NOTE: Available at https://infoscience.epfl.ch/record/277809.

[i.9] Ghazaleh Beigi, Huan Liu: "A Survey on Privacy in Social Media: Identification, Mitigation, and Applications". ACM/IMS Transactions on Data Science, March 2020, Article No. 7.

NOTE: Available at https://doi.org/10.1145/3343038.

[i.10] IETF RFC 8446: "The Transport Layer Security (TLS) Protocol Version 1.3".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

diagnosis key: secret code from which ephemeral identifiers for a given period of time can be derived with the help of a cryptographic function

ephemeral identifier: unique device identifier exchanged with another device during a proximity event

proximity event: event recorded by the software on a mobile device corresponding to proximity to other device with an active interoperable applications, and which meets the predefined criteria for an event to be recorded (e.g. duration)

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ACK	Acknowledgement
BCD	Requirement "Back-end Centralized Diagnosed"
BCE	Requirement "Back-end Centralized EBID"
BCP	Requirement "Back-end Centralized Privacy"
BCR	Requirement "Back-end Centralized Registration"
BCS	Requirement "Back-end Centralized Setup"
BCX	Requirement "Back-end Centralized Exposure"
BDB	Requirement "Back-end Decentralized BaseURL"
BDD	Requirement "Back-end Decentralized Data-Structures"
BDK	Requirement "Back-end Decentralized Key Download"
BDO	Requirement "Back-end Decentralized One-Time-Token"
BDP	Requirement "Back-end Decentralized Parameters"
BDR	Requirement "Back-end Decentralized Retrieve Test Results"
BDT	Requirement "Back-end Decentralized TLS"
BDU	Requirement "Back-end Decentralized Key Uplead"
BE	Reference Point "Back-end - External System"
BF	Reference Point "Back-end - Federation"
Bluetooth® LE	Bluetooth Low Energy
CDN	Content Distribution Network
DB	Reference Point "Device - Back-end"
DCTS	Digital Contact Tracing System
DP3T	Decentralized Privacy-Preserving Proximity Tracing
EBID	Ephemeral Bluetooth® IDentifier
ESR	Exposure Status Request
GAEN	Google Apple Exposure Notification
GDPR	General Data Protection Regulation
GUID	Globally Unique Identifier
HSM	Hardware Security Module
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JSON	JavaScript Object Notation
MB	Mbits/s
NTP	Network Time Protocol
PKI	Public Key Infrastructure
QR	Quick Response
TAN	Transaction Authentication Number

TEE	Trusted Execution Environment
TLS	Transport Layer Security
URL	Uniform Resource Locator
UTF	Unicode Transformation Format

4 Definition and Extent of Back-End Mechanisms

4.1 Mapping to Reference Model from Requirements Document

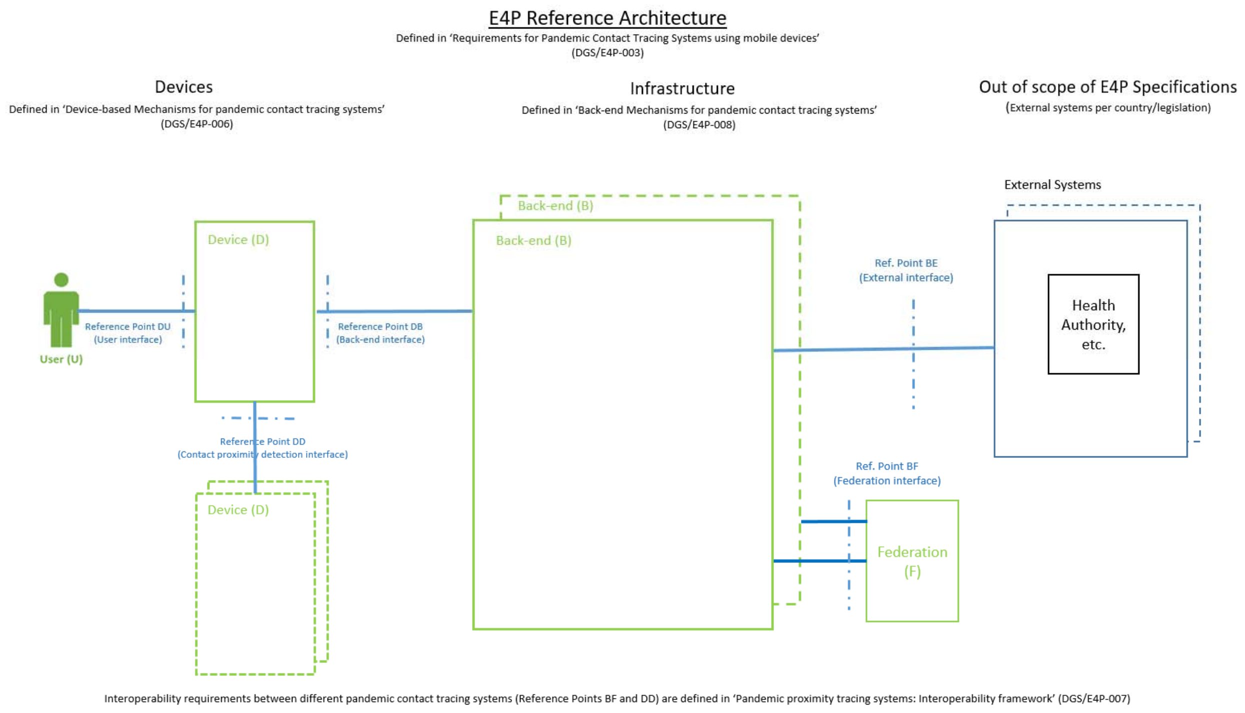
E4P Reference Architecture
Defined in 'Requirements for Pandemic Contact Tracing Systems using mobile devices' (DGS/E4P-003)
Devices
Defined in 'Device-based Mechanisms for pandemic contact tracing systems' (DGS/E4P-006)
Infrastructure
Defined in 'Back-end Mechanisms for pandemic contact tracing systems' (DGS/E4P-008)
Out of scope of E4P Specifications
(External systems per country/legislation)
External Systems
Device (D)
User (U)
Reference Point DU (User interface)
Reference Point DB (Back-end interface)
Back-end (B)
Back-end (B)
Ref. Point BE (External interface)
Health Authority, etc.
Reference Point DD (Contact proximity detection interface)
Device (D)
Ref. Point BF (Federation interface)
Federation (F)
Interoperability requirements between different pandemic contact tracing systems (Reference Points BF and DD) are defined in 'Pandemic proximity tracing systems: Interoperability framework' (DGS/E4P-007)

Figure 1: E4P Reference Architecture

The present document covers all back-end mechanisms necessary to make a digital contact tracing system functional. Depending on the nature of the system, certain functions reside in the central infrastructure or in the device.

Functions west-bound from the reference point "DB" are covered in ETSI GS E4P 006 [2].

Mechanisms that facilitate interoperability between digital contact tracing systems, east-bound from the reference point "BF", like a federation function between systems using the same design approach, or between systems with different design approaches, are addressed in ETSI GS E4P 007 [3].

Digital contact tracing systems need to be integrated into a regional or national regime of handling a pandemic. Functions in this context can be e.g. testing or setting of parameters to calibrate or adapt the system. These functions are eastbound of the reference point "BE" and out of scope of the ISG E4P.

4.2 Systems and components out of scope

Deployment of a Digital Context Tracing System (DCTS) requires a suitable technical and organizational integration with regional or national authorities and health entities, part of or working under authorization of the public health authorities (in the scope of overall contact tracing initiated by public health entity/ies).

DCTS provides the technical means to securely measure, collect, calculate, and communicate data according to and in the context of an overall scheme defined by epidemiological and legal experts that have the respective scientific and formal authorizations. The definition of the parameters to tune a DCTS, e.g. distance and time of proximity to be measured, risk calculation algorithms, legal requirements on data handling, etc. are defined and decided on outside of the DCTS. The E4P Reference Architecture refers to this complex as "External Systems" and defines the Reference Point BE (Back-End - External Systems) for the respective information flows.

Figure 2 shows some examples of information flow types initiated by External Systems that might or might not be handled by the Infrastructure as defined in the present document. If the information flow handled by the Infrastructure the information has to make use of the Reference Point BE. An example from the present document would be clause 6.1.6.

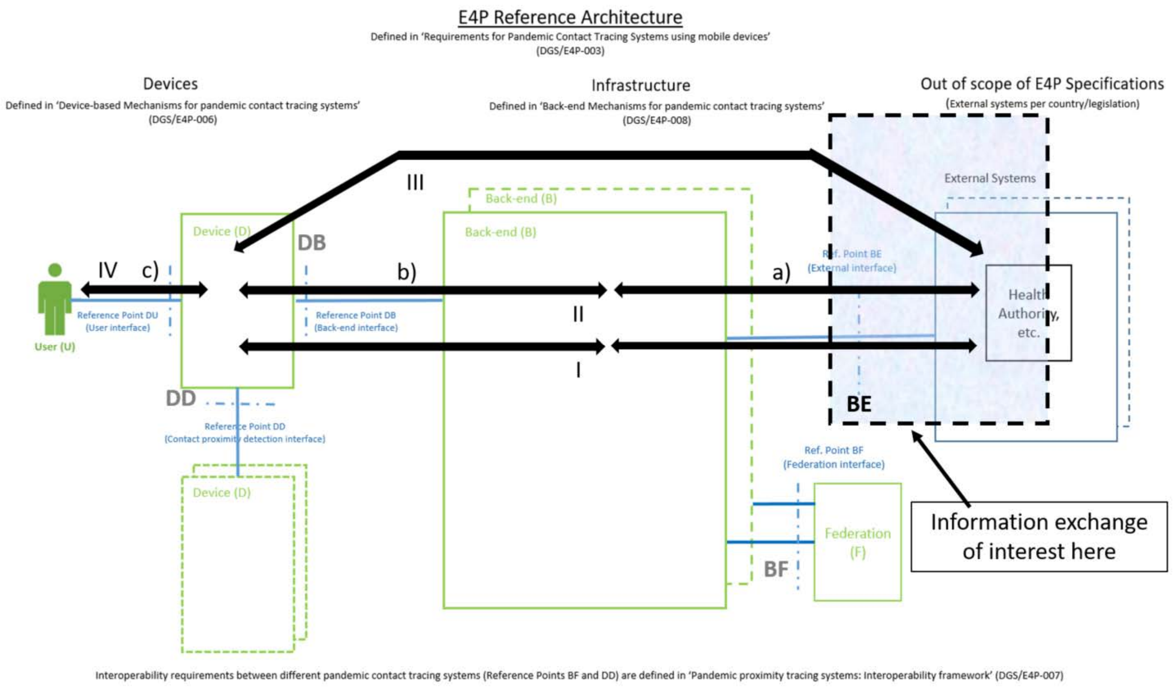
E4P Reference Architecture
Defined in 'Requirements for Pandemic Contact Tracing Systems using mobile devices' (DGS/E4P-003)
Devices
Defined in 'Device-based Mechanisms for pandemic contact tracing systems' (DGS/E4P-006)
Infrastructure
Defined in 'Back-end Mechanisms for pandemic contact tracing systems' (DGS/E4P-008)
Out of scope of E4P Specifications
(External systems per country/legislation)
III
External Systems
Back-end (B)
Device (D)
DB
Back-end (B)
Ref. Point BE (External interface)
IV
c)
b)
a)
Health Authority, etc.
Reference Point DU (User interface)
User (U)
Reference Point DB (Back-end interface)
II
I
DD
BE
Reference Point DD (Contact proximity detection interface)
Ref. Point BF (Federation interface)
Device (D)
Federation (F)
Information exchange of interest here
BF
Interoperability requirements between different pandemic contact tracing systems (Reference Points BF and DD) are defined in 'Pandemic proximity tracing systems: Interoperability framework' (DGS/E4P-007)

Figure 2: E4P High-level Reference Architecture indicating different types of information flows with External Systems

"Receive Test-Result" is informally represented in Figure 2 by the arrows II indicating the steps:

a) test result received by the Back-end;

b) test result retrieved by the Device;

c) test result displayed and acted on by the user.

In case of a positive test result, the release of 'diagnosis keys' is initiated in one of two ways: by an action by the user or by an action by the App software on the device; in the latter case the DCTS App informs the user.

The arrow I represent an information flow of the type "Configuration-Parameter Download", where the parameters affect the functions performed by the device, but are not necessarily displayed to the user.

The arrow III represents an "out-of-band" flow, where information to the user is bypassing the DCTS infrastructure.

The arrow IV represents all interactions between the users and their smartphones in relation with DCTS; this includes interactions with the DCTS App, but may include interactions with other Apps, e.g. a browser or e-mail client used to obtain useful information related to the use of DCTS.

In case of the decentralized approach the downloaded Diagnosis Keys come together with a digital signature to allow the Mobile Device to verify, that the Diagnosis Keys are genuine and were not manipulated during transport. The concrete way how the Mobile App gets the signature test key is out of scope of the present document (see clause 6.1.3).

In clause 6.1.6, the reception of test results from the responsible authority shall be authenticated. How this authentication is done is out of scope of the present document.

In clause 6.1.7, it is out of scope of the present document how the health authority decides to issue a teleTAN in the case a laboratory does not support the procedure described in this clause.

In clause 6.1.8, the details regarding the configuration parameters to alter/tune the risk calculation algorithm are out of scope of the present document.

5 Back-end mechanisms for pandemic contact tracing systems

5.1 Decentralized approach

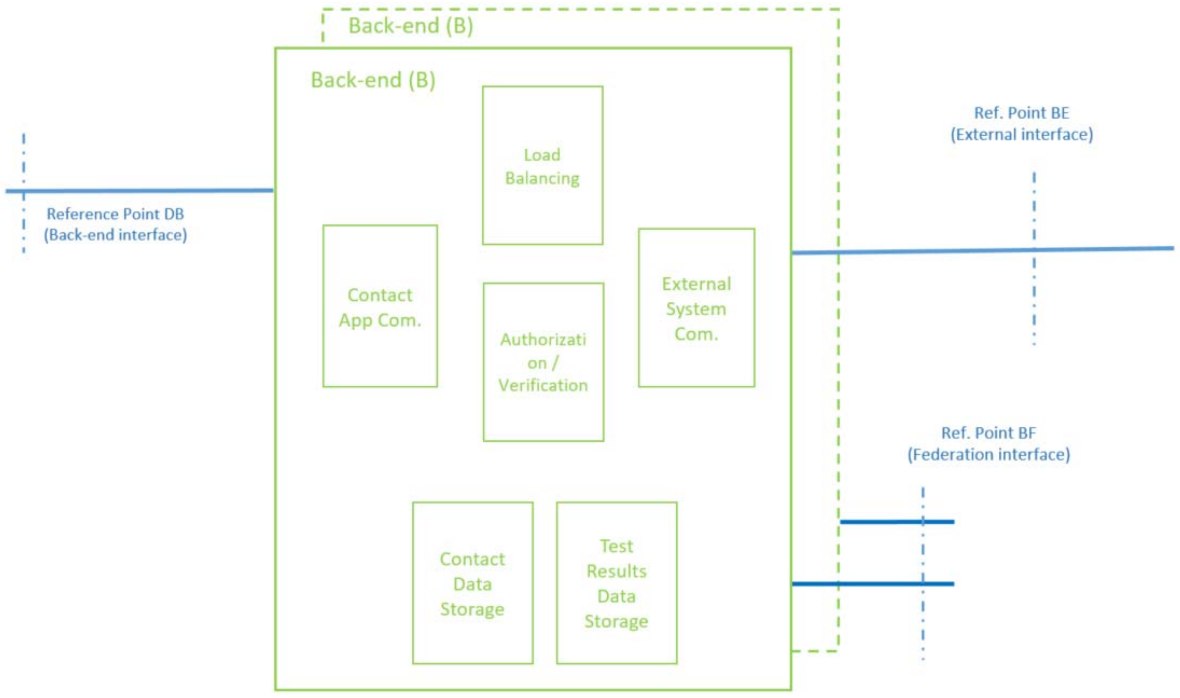
Back-end (B)
Back-end (B)
Load Balancing
Contact App Com.
Authorization / Verification
External System Com.
Contact Data Storage
Test Results Data Storage
Reference Point DB (Back-end interface)
Ref. Point BE (External interface)
Ref. Point BF (Federation interface)

Figure 3: Back-end mechanisms overview (decentralized)

External System Communication

The External System Communication component interacts with external systems (e.g. Health Authorities, Test Laboratories) in order to acquire or provide necessary information (e.g. test results, parameters to tune the overall system or the exposure calculation formula, etc.) to make the overall Contact Tracing System functional.

NOTE 1: The E4P Reference Architecture also shows a Reference Point BF towards functions or modules realizing federation between two or more Digital Contact Tracing Systems. However, no federation specific functions inside a Back-end are described in the present document. For details regarding federation see ETSI GS E4P 007 [3].

Contact App Communication

The Contact App Communication component interacts with the Mobile Device to Upload Diagnosis Keys (e.g. if User is diagnosed) and to download periodically the Diagnosis Keys (needed to calculate the Users exposure to other diagnosed Users) and parameters to tune the risk calculation. In the latter case a Load Balancing Function may be used to cope with the number of mobile devices. The Contact App Communication component is also used to perform some "dummy" communication between Mobile Devices and the Back-end to make the Upload unobservable.

Authorization/Verification

The Authorization/Verification component authorizes specific Users for specific actions (e.g. Uploads) by appropriate means (e.g. one-time PINs) and verifies the correct authorizations in the respective steps.

Contact Data Storage

The Contact Data Storage component stores the sets of Diagnosis Keys the DCTS has to handle and provides access to them according to the specified protocols and rules.

Test Result Data Storage

The Test Result Data Storage component stores the test results necessary to forward to Users and provides access only to verified authorizations.

Load Balancing (e.g. CDN)

The Load Balancing component is an optional function that replicates needed resources in order to guaranty maximum availability of the DCTS. Below are some illustrative calculations regarding the expected amount of data which needs to be transmitted to each Mobile Device in case of a decentralised DCTS.

NOTE 2: The following calculation is based on GAEN [i.1].

In the decentralised system design each Mobile Device locally checks if the device was in close proximity to other mobile devices of diagnosed users. Therefore, each Mobile Device needs to know all the relevant diagnosis keys from diagnosed users. In the current system design "relevant keys" usually refers to the diagnosis keys used by an diagnosed user during 14 days (depending on the epidemiological necessities, e.g. at least 14 days for COVID-19) prior to the point in time at which the user uploads his/her diagnosis keys to the back-end.

Moreover, the diagnosis keys change on a daily base, i.e. there is one diagnosis key per mobile device per day. Taking this as input for the calculations and assuming that each user uses only one mobile device, each newly diagnosed user will upload 14 diagnosis keys to the back-end, which need to be distributed to all other mobile devices.

A single diagnosis key consists of 16 random bytes. Therefore, each newly diagnosed user contributes at least $14 \times 16 = 224$ random bytes to the amount of data which need to be distributed to each other mobile device per day. Note that because of the random nature of these bytes they cannot be compressed. Assuming 10 000, 100 000 or 1 000 000 newly diagnosed users this will lead to an amount of data of 2,24 MB, 22,4 MB or 224 MB respectively.

Encoding the diagnosis keys using appropriated data structures will require some extra bytes. Given that the protocol buffer data structure `TemporaryExposureKey` as describe in clause 6.1.1 is used for encoding diagnosis keys, the additional overhead will be roughly 11 bytes per diagnosis key.

These bytes can be compressed making an exact calculation of the amount of data traffic necessary to transmit the diagnosis keys difficult. In a worst case scenario it would need $14 \times (16 + 11) = 378$ bytes per newly diagnosed user. So, for 10 000, 100 000 or 1 000 000 newly diagnosed users this will lead to an amount of data of 3,78 MB, 37,8 MB or 378 MB respectively.

5.2 Centralized approach

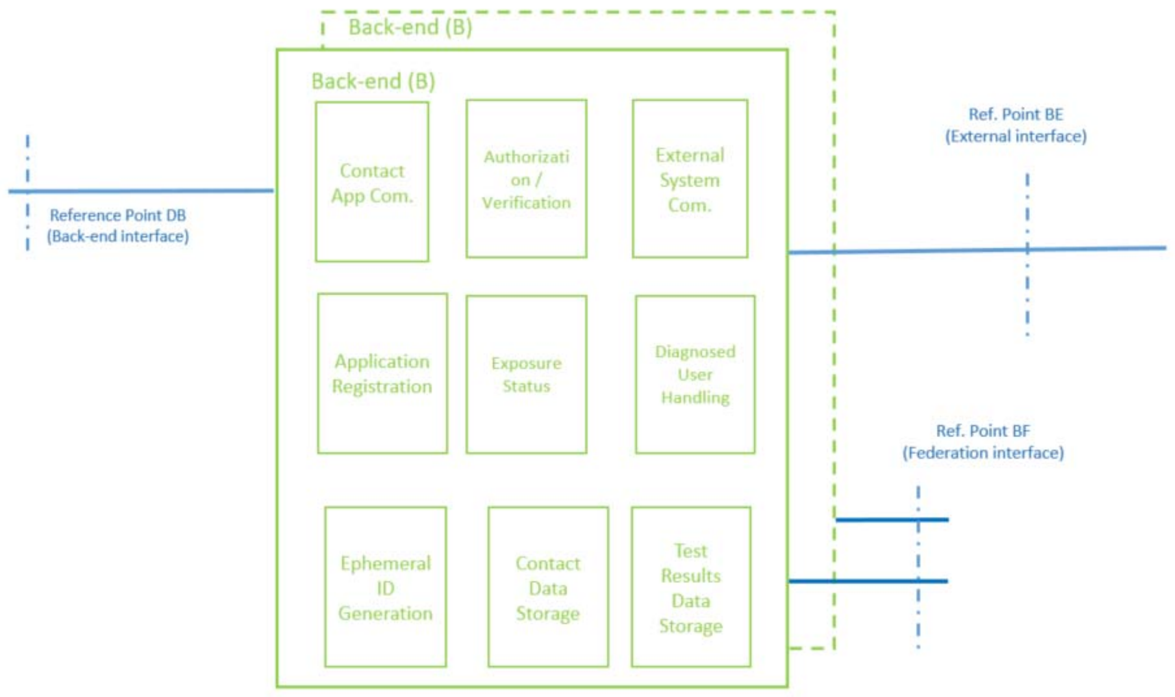

Figure 4: Back-end mechanisms overview (centralized)

Application registration

The Application registration component is responsible for performing all the operations that are needed when a Mobile Application registers to the Back-end (the first time a newly activated Mobile Application contacts the Back-end).

Ephemeral Ids generation

The Ephemeral Ids generation component provides the generation of the ephemeral identifiers that will be used by each Mobile Device. These ephemeral identifiers are regularly sent by the Back-end to the Mobile Application.

Diagnosed user handling

The Diagnosed user handling component handles the necessary operations when a User is diagnosed. In this case, the Mobile Application downloads what is essentially the relevant ephemeral identifiers that the Mobile Application has seen during the period within which the User may have be contagious.

Exposure status

The Exposure status component is responsible for determining whether the User of a given Mobile Device is considered as at risk for having been in close proximity (according to a risk calculation method) with another diagnosed User (this is done by comparing the ephemeral identifiers that were downloaded to the Back-end via the Diagnosed user handling function to the ephemeral identifiers that were generated for the User via the Ephemeral Ids generation function). The Mobile Application regularly polls the Back-end to check whether its User should be considered at risk.

6 Information flows, protocols and Data Structures

6.1 Decentralized approach: GAEN

6.1.0 Overview

The back-end of a decentralized DCTS is mainly an information hub collecting and providing information to other systems without having to understand or being able to make use of the data handled in any way that would compromise the privacy of the Users.

The main tasks of the Back-end are:

- Handling of Diagnosis Keys, both from and to the Mobile Devices.
- Handling of test results received from e.g. Test Laboratories and making them available to the respective Users.
- Handling of parameters to tune the DCTS, as issued by the Health Authority.

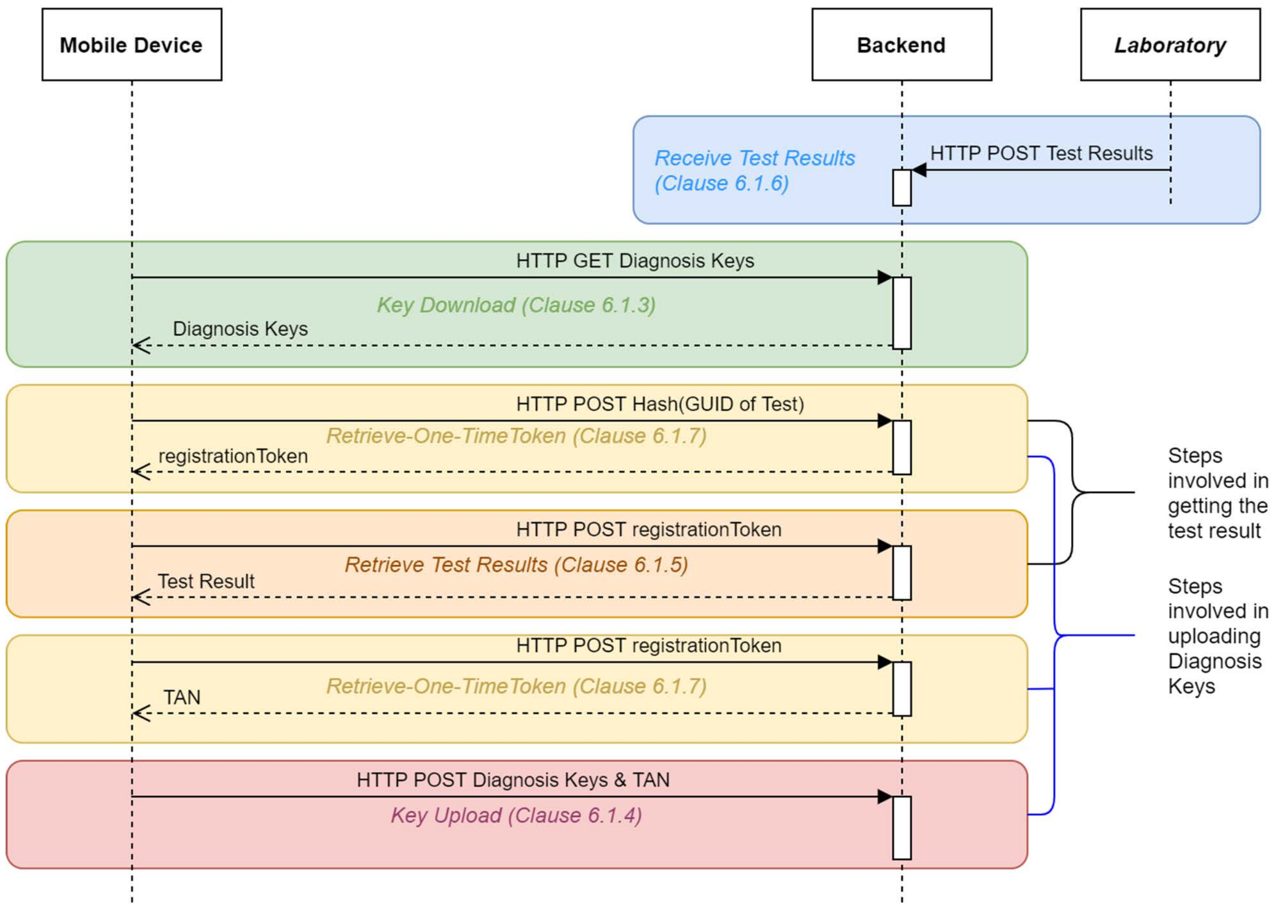
Mobile Device
Backend
Laboratory
Receive Test Results (Clause 6.1.6)
HTTP POST Test Results
HTTP GET Diagnosis Keys
Key Download (Clause 6.1.3)
Diagnosis Keys
HTTP POST Hash(GUID of Test)
Retrieve-One-TimeToken (Clause 6.1.7)
registrationToken
HTTP POST registrationToken
Retrieve Test Results (Clause 6.1.5)
Test Result
HTTP POST registrationToken
Retrieve-One-TimeToken (Clause 6.1.7)
TAN
HTTP POST Diagnosis Keys & TAN
Key Upload (Clause 6.1.4)
Steps involved in getting the test result
Steps involved in uploading Diagnosis Keys

Figure 5: DCTS Back-end information flows (decentralized)

6.1.1 Generic Data Structures

[BDD-01] The following data structures describe protocol buffer data structures initially defined by GAEN [11] used during the transfer of Diagnosis Keys. They shall be used as building blocks used to construct the actual data structures used during Key Download and Key Upload.

```
message TemporaryExposureKey {
    // The actual Diagnosis Key of an diagnosed user
  optional bytes key_data = 1;
    // The interval number since epoch for which a key starts
  optional int32 rolling_start_interval_number = 2;
    // Increments of 10 minutes describing how long a key is valid
  optional int32 rolling_period = 3
      [default = 144]; // defaults to 24 hours
     // Data type representing why this key was published.
  enum ReportType {
    UNKNOWN = 0;
    CONFIRMED_TEST = 1;
    CONFIRMED_CLINICAL_DIAGNOSIS = 2;
    SELF_REPORT = 3;
    RECURSIVE = 4;  // Reserved for future use.
    REVOKED = 5;  // Used to revoke a key.
  }

    // Type of diagnosis associated with a key.
  optional ReportType report_type = 5;

    // Number of days elapsed between symptom onset and the Diagnosis Key
    // being used.
    // E.g. 2 means Diagnosis Key is used 2 days after onset of symptoms.
  optional sint32 days_since_onset_of_symptoms = 6;
}
```

Protocol Buffer Data Structure describing a single Diagnosis Key.

```
message SignatureInfo {
    // Signature test key version for rollovers
    // Must be in character class [a-zA-Z0-9_]. For example, 'v1'
  optional string verification_key_version = 3;
    // Alias with which to identify the Signature test key used for verification
    // Must be in character class [a-zA-Z0-9_.]
  optional string verification_key_id = 4;
    // ASN.1 OID for Algorithm Identifier.
  optional string signature_algorithm = 5;
}
```

Protocol Buffer Data Structure containing information for referencing to a key which can be used to verify a given digital signature.

```
message Signature {
    // Information about the signing key, version, algorithm, and so on.
  optional SignatureInfo signature_info = 1;
    // Describes for which part this signature is
    // Number of the part
  optional int32 batch_num = 2;
    // Total number of parts
  optional int32 batch_size = 3;
    // The actual bytes of the digital signature
    // The specific format depends on the signature algorithm
   optional bytes signature = 4;
}
```

Protocol Buffer Data Structure describing a single digital signature.

6.1.2 End-to-End Transport Layer Encryption

All communication between the Mobile Device and the Back-end shall be secured by an End-to-End Transport Layer Encryption.

[BDT-01] The backend shall at least support TLS version 1.2 as specified in IETF RFC 5246 [4]. The backend shall not support any version of TLS below TLS version 1.2. It is recommended to use TLS version 1.3 as specified in IETF RFC 8446 [i.10].

[BDT-02] The backend shall only support cipher suites which offer a security level equivalent to 112 bit symmetric key encryption. The backend shall support cipher suites which offer Perfect Forward Secrecy. The Mobile Device should use a cipher suite which implements Perfect Forward Secrecy for connecting with the backend.

6.1.3 Key Download

This clause describes the information flows and protocols for providing the Diagnosis Keys Download function. There is no access control for downloading the Diagnosis Keys, since it is a fundamental design principle to broadcast the Diagnosis Keys to any user of the decentralised DCTS.

[BDK-01] The Diagnosis Keys Download function shall provide the Diagnosis Keys in the Exposure Key export file format as described by GAEN [11]. Each provided file is a zip [8] archive containing exactly two entries:

`export.bin` and `export.sig`.

NOTE 1*:* A zip archive is used here because it allows to store the two entries within a single file. The usually provided compression functionality is of no use here since the main part of the content of the zip archive are random bytes (Diagnosis Keys).

The entry `export.bin` stores the Diagnosis Keys *received* by the Backend during a certain time frame (typically a whole day) utilizing a protocol buffer data structure. Grouping Diagnosis Keys according to the receive time would make incremental updates for a Mobile Device easier. The ordering of the Diagnosis Keys shall not allow to draw any conclusion with respect to the original order of reception of Diagnosis Keys by the back-end with respect to the related time interval.

```
message TemporaryExposureKeyExport {
    // The two timestamps below describe the times during which the contained
    // keys arrived at the backend (UTC seconds).
  optional fixed64 start_timestamp = 1;
  optional fixed64 end_timestamp = 2;
    // Describes the region from which these Diagnosis Keys came from.
  optional string region = 3;
    // The following two entries can be used to split the whole Diagnosis Key
    // export over multiple files.
    // Number of the part of the keys contained in this file
  optional int32 batch_num = 4;
    // Total number of parts
  optional int32 batch_size = 5;
    // Information about the signatures used to sign this data structure
  repeated SignatureInfo signature_infos = 6;
    // The actual (initially uploaded) Diagnosis Keys
  repeated TemporaryExposureKey keys = 7;
    // Diagnosis Keys which changed their status since the last upload
  repeated TemporaryExposureKey revised_keys = 8;
}
```

Protocol Buffer Data Structure within the `export.bin` entry of the Key Download HTTP GET request.

The `export.bin` binary file consist of a header followed by the binary representation of the protocol buffer data structure described above. The header has the following content, which is padded by spaces encoded in UTF-8 (IETF RFC 3629 [6]) 16 bytes in total:

```
EK Export v1
```

Header of `export.bin`

Therefore, the bytes of the header are:

```
0x45 0x4B 0x20 0x45 0x78 0x70 0x6F 0x72 0x74 0x20 0x76 0x31 0x20 0x20 0x20 0x20
```

Binary representation of the header of `export.bin`

The entry `export.sig` stores a digital signature over the `export.bin` entry. This allows the Mobile Device to verify, that the Diagnosis Keys are genuine and were not manipulated during transport (e.g. in case a Content Distribution Network (CDN) is used). The Mobile Device shall verify this digital signature with the help of the corresponding public key. A Mobile Device shall only process an `export.bin` entry, if and only if the signature could be successfully verified. The public signature test key can be either part of the Mobile App or could be downloaded from the backend or some other trustworthy place e.g. belonging to a Public Key Infrastructure (PKI). The concrete way how the Mobile App gets the signature test key is out of scope of the present document.

The entry `export.sig` is a serialized form of the following protocol buffer data structure:

```
message SignatureList {
    //The actual digital signatures for the corresponding export.bin entry
  repeated Signature signatures = 1;
}
```

Protocol Buffer Data Structure within the `export.sig` entry of the Key Download HTTP GET request.

NOTE 2: This application layer signature would allow the Mobile Device to verify the validity of the provided Diagnosis Keys independent of any authentication mechanism the transport layer may provide.

The zip archives for the different time periods containing the corresponding Diagnosis Keys shall be provided to the Mobile Device by means of the Hypertext Transfer Protocol (see IETF RFC 2616 [5]). In order to download a single zip archive a Mobile Device shall issue a HTTP GET request. The URL of that request shall encode in which zip archive the Mobile Device is interested in.

The URL pointing to a given zip file should be constructed as described in the following paragraphs.

[BDB-01] There shall be a base URL (`baseURL`) which serves as a starting point for creating the final download URL. This `baseURL` shall be known to the Mobile Device.

The Key Download procedure should support different versions, e.g. for allowing future updates of the present document. Therefore, a path encoding the version of the Key Download procedure should follow the `baseURL`. The resulting `versioned_baseURL` will be as follows: `baseURL/version/VERSIONTAG`. The only valid value for the `VERSIONTAG` is `v1`, i.e. the `versioned_baseURL` will be: `baseURL/version/v1`.

The URL for downloading the keys encodes the country of origin of the corresponding keys. In order to get a list of countries for which the backend provides diagnosis keys an HTTP GET request with the accepted content type set to `application/json` and the following URL should be used: `versioned_baseURL/diagnosis-keys/country`. The returned JSON data structure shall contain an array of the country codes for which diagnosis keys are stored.

In order to download diagnosis keys regarding a specific country of origin the following `versioned_country_baseURL` is defined: `versioned_baseURL/diagnosis-keys/country/COUNTRYCODE`. Moreover, to learn for which dates diagnosis keys are stored in the backend an HTTP GET request with the accepted content type set to `application/json` and the following URL should be used: `versioned_country_baseURL/date`. The returned JSON data structure shall contain an array of the dates for which diagnosis keys are stored.

[BDB-02] The final download URL for diagnosis keys with respect to a given country of origin and date shall be constructed as follows: `versioned_country_baseURL/date/DATE`. An example download URL may look as follows: `https://download.backend.server/version/v1/diagnosis-keys/country/DE/2020-11-22`

6.1.4 Key Upload

This clause describes the information flows and protocols for providing the Diagnosis Keys Upload function.

[BDU-01] The Diagnosis Keys of a User shall be transferred to the Back-end by means of a HTTP POST request. The payload of this HTTP POST request shall be a protocol buffer data structure (named `SubmissionPayload` below) and shall contain at least the actual keys and some meta data as described below:

```
message SubmissionPayload {
   // The Diagnosis Keys of an diagnosed user
  repeated TemporaryExposureKey keys = 1;
  // if set to TRUE, the diagnosed user has agreed
  // that these Diagnosis Keys can be forwarded to other backends
  // for the purpose of interoperability (federation)
  optional bool consentToFederation = 2;
}
```

Protocol Buffer Data Structure for the payload of the Key Upload HTTP POST request.

[BDU-02] The Backend shall only accept authorized Uploads of Diagnosis Keys. Therefore, the HTTP POST request shall contain the necessary credentials, which allow the Back-end to verify, that the Key Upload is authorized.

[BDU-03] The Backend should support at least a TAN-based authorization procedure. The procedures to get a TAN based on other credentials in possession of a user (valid positive test results, etc.) are described in clause 6.1.7. If the TAN-based authorization is used, the HTTP POST request shall contain an additional HTTP header which contains the TAN to authorize the Key Upload as described below:

```
E4P-Submission-Authorization-TAN: <TAN>
```

HTTP header used to transport a TAN for authorizing this Key Upload.

As mentioned above the communication shall be secured by end-to-end encryption at the transport layer.

[BDU-04] To further enhance the privacy of diagnosed users, the backend shall support the upload of dummy keys. The upload of dummy keys shall be indistinguishable by external attackers from the upload of real keys. Therefore, the message flows, payload sizes, timings, etc. need to be like the corresponding properties of real key uploads. It is allowed, that the Back-end itself distinguishes real keys from dummy keys. Therefore, each HTTP POST request shall contain an additional HTTP header which indicates if the payload contains real keys or dummy keys. This HTTP header is described below:

```
E4P-Submission-Dummy: <Either 0 or 1>
```

HTTP header used to indicate if the payload contains real or dummy keys. The value 0 indicates, that the payload contains real keys, the value 1 indicates, that the keys are dummies.

6.1.5 Retrieve Test Results

In order to retrieve test results stored in the back-end a Mobile Device shall prove, that it is actually allowed to retrieve the requested test result. Therefore, the Mobile Device shall present a so-called registrationToken to the backend servers. A Mobile Device can get a registrationToken from the back-end with the help of the Retrieve-On-Time-Token function (see clause 6.1.7). A given registrationToken is only valid for retrieving the result of one specific test. This test is uniquely identified by the registrationToken, therefore it is sufficient, if a Mobile Device submits the registrationToken to the back-end to get the test result. Retrieving test results is a pull-based mechanism, i.e. a Mobile Device needs to query the back-end from time to time to receive new test results.

In order to get the test result, the Mobile Device shall send the registrationToken to the backend using a HTTP POST request. The payload of this request shall contain the registrationToken encoded in a JSON data structure as follows:

```
{"registrationToken": "THE-VALUE-OF-THE-REGISTRATION-TOKEN"}
```

[BDR-01] The payload of the HTTP response shall contain the result of the related test encoded in a JSON data structure. The following test results are possible (encoded by the number given):

```
0 – no test result available yet

1 – negative test result

2 – positive test result

3 – test result is invalid (reason not further specified)
```

The JSON data structure of the response payload is as follows:

```
{"testResult":THE-TEST-RESULT}
```

[BDR-02] To enhance privacy, the backend shall support dummy requests for retrieving test results. The `E4P-Dummy` HTTP header (see clause 6.1.4) can be used to indicate, if a request is a dummy request.

[BDR-03] The backend should respond to a dummy request in a way which is indistinguishable from a response to a real request with respect to external attackers.

6.1.6 Receive Test Results

Test results are submitted by external entities, usually laboratories. Every test can be uniquely identified by a globally unique identifier (GUID). Each test result can have one of the following states:

```
1 – negative test result

2 – positive test result

3 – test result is invalid (reason not further specified)
```

[BDR-04] In order to submit a test result, the laboratory shall provide a hash of the GUID together with the actual test result. SHA-256 [10] shall be used to generate the hash of the GUID. The test result shall be submitted by issuing a HTTP POST request. In order to make the interface more efficient, an external entity can submit multiple test results within one HTTP POST request. Therefore, a list of test results can be transmitted. The payload of this HTTP POST request shall contain the list of test results encoded in a JSON data structure as described below:

```
{"testResults":
    [
        {"id" : "HASH-OF-GUID-OF-TEST" , "result" : THE-TEST-RESULT},
        {"id" : "HASH-OF-GUID-OF-TEST" , "result" : THE-TEST-RESULT},
        …
    ]
}
```

[BDR-05] This HTTP POST request shall be authenticated to ensure, that it originated in fact from an entity authorized to submit test results. How this authentication is done is out of scope of the present document.

[BDR-06] Moreover, the connection between the external entity and the backend shall be secured by means of TLS as described in clause 6.1.1.

6.1.7 Retrieve One-Time-Token

The Retrieve-One-Time-Token function allows the Mobile Device of the User to get a so-called registrationToken and a TAN. The registrationToken is used to retrieve the test result from the backend (see clause 6.1.5) and the TAN is used to authorize the upload of the Diagnosis Keys (see clause 6.1.4).

In order to retrieve the test result, an identifier (GUID) of the test known to the user is converted into a registrationToken. The test identifier known to the user is actually a number printed on the paper-based recipe the user gets in the course of the test procedure. Usually, the number is printed as a QR code making it more convenient for the user to enter the test number into the Mobile App e.g. by letting the Mobile Device scan the QR code.

For privacy reasons the test identifier is not directly used for retrieving the test result. Since it may take some time before the laboratory has finished the test and uploaded the test result to the backend the test identifier would need to be stored on the Mobile Device. This induces a privacy risk if the Mobile Device gets compromised or gets lost or stolen. Therefore, the Mobile Device shall not store the test identifier. Storing a hash value of the test identifier would not solve the privacy issue, since an attacker could check, if a given test belongs to a given user. In fact the conversion of the test identifier into the registrationToken mitigates this privacy risk, since the registrationToken can only be linked to a given test by the back-end.

[BDO-01] The back-end shall not store the test identifier directly but shall store only a hash of the test identifier for enhanced security.

A second purpose of the registrationToken is to get a TAN, if the related test indicates a positive test result. Therefore, the registrationToken is submitted to the back-end, the back-end checks if the related test result is positive and issues a TAN. While the registrationToken is a rather long lasting credential, the only purpose of the TAN is the authorization of the upload of the Diagnosis Keys.

[BDO-02] The TAN shall be only short-lived and shall be deleted if it was successfully used for Key Upload or within a reasonable number of hours after creation, depending on the overall system setup. The conversion of the registrationToken into a TAN adds an additional layer of decoupling of the test from the Key Upload. This could especially be a means for enhanced privacy, if there is an organizational separation between the operation of the Retrieve-One-Time-Token function and the Key Upload function.

Not in all cases the user receives a test identifier, e.g. if the laboratory does not yet support this procedure. Since even in this case a positive tested user should be able to submit his/her diagnosis keys, additional means for authorizing a key upload are necessary. One solution is, that the health authority authorizes the Key Upload by providing the user with a so-called teleTAN. How the health authority decides to issue a teleTAN is out of scope of the present document.

One possible reason could be, that the health authority gets aware of the positive test result of a given user in the course of manual contact tracing. To make the teleTAN-based authorization of the Key Upload as much as possible indistinguishable from the test identifier-based authorization of the Key Upload, the procedures are unified. Therefore, the teleTAN is converted into a registrationToken and the registrationToken is converted into a TAN.

If the teleTAN is stored in the back-end (or more specific a hashed value of the teleTAN) it shall be deleted not later than 1 hour after creation.

[BDO-03] In order to get a registrationToken, the Mobile Device shall issue a HTTP POST request.

[BDO-04] The payload of this request shall be a JSON data structure which contains either a hash of a test identifier or a teleTAN and specifies which of the two is contained. The JSON data structure is as follows:

```
{
  "key":"the value of test identifier or teleTAN",
  "keyType":"TELETAN" | "HASHEDGUID"
}
```

The `keyType` encodes, if the value given for the `key` is actually a hash of a test identifier (keyType=HASHEDGUID) or a teleTAN (keyType=TELETAN). SHA-256 [10] shall be used for creating the hash of the test identifier.

[BDO-05] The payload of the response shall be a JSON data structure containing a registrationToken as described below:

```
{
  "registrationToken":"the value of the registration token"
}
```

[BDO-06] The backend shall ensure, that for every test identifier and every teleTAN not more than one registrationToken can be retrieved.

[BDO-07] The backend shall verify, if a provided teleTAN is valid.

In order to get a TAN, the Mobile Device shall issue a HTTP POST request.

[BDO-08] The payload of this request shall be a JSON data structure which contains a registrationToken. This JSON data structure is as follows:

```
{
  "registrationToken":"the value of the registration token"
}
```

[BDO-09] The payload of the response shall be a JSON data structure containing a TAN as described below:

```
{
  "tan":"the value of the TAN"
}
```

[BDO-10] The back-end shall only return a TAN, if the registrationToken refers to either a positive test result (in case the registrationToken originates from a test identifier) or if the registrationToken refers to a valid teleTAN (in case the registrationToken originates from a teleTAN).

[BDO-11] The backend shall ensure that for every valid registrationToken not more than one TAN can be retrieved.

[BDO-12] The backend shall never store registrationTokens, teleTANs or TANs. If necessary, the backend can store hashed versions of them.

[BDO-13] In order to enhance privacy, the backend shall support dummy requests for retrieving a registrationToken and for retrieving a TAN. The `E4P-Dummy` HTTP header (see clause 6.1.3) can be used to indicate if a request is a dummy request.

[BDO-14] The backend should respond to a dummy request in a way which is indistinguishable from a respond to a real request with respect to external attackers.

6.1.8 Configuration-Parameters Download

The back-end can provide a function, which allows the Mobile App to retrieve some configuration parameters e.g. related to the risk score calculation.

In order to retrieve the configuration parameters, the Mobile Device should issue a HTTP GET request.

[BDP-01] The payload of the response should contain the configuration parameters. These configuration parameters shall be authenticated with the help of one or more digital signatures. Therefore, the payload can be of a similar format as the one used for Key Download, i.e. it can be a zip file, which shall contain two files: `export.bin` and `export.sig`.

In this case, the file `export.`sig shall contain one or more digital signatures which shall be used to verify the validity of the `export.bin` file. The `export.bin` file should be a protocol buffer data structure containing the actual configuration parameters. The details regarding the configuration parameters contained within the file `export.bin` are specific to a given Mobile Application and are out of scope of the present document.

6.2 Centralized approach

6.2.1 Server set up

This clause describes in some details the ROBERT solution (see [i.3] for more details).

The back-end server is initialized at the beginning of the proximity tracing official period under the control of the Health Authority. In order to be able to determine in which period and epoch the system is, the server stores $T_{ptsstart}$, the time when the proximity tracing service has been started in the country. From this timestamp, the server maintains a counter i, initialized to 0, representing the current epoch number coded by 24 bits.

If the server is configured with:

- K_S ("Server Key"): a L-bit long key, with L >= 128 to be defined, only known by the server.
- K_G ("Federation Key"): a L-bit long key, with L >= 128 to be defined, shared between all servers in a Federation.
- (sk_S, pk_S) ("Registration key-pair"): an asymmetric-key pair, whose private key sk_S is known only to the server, and public key part is known to every App. This key-pair is defined over the elliptic curve NIST-P256, with $pk_S = sk_S.G$, where G is the base point of prime order on the curve defined by the specifications [9].
- CC_S ("Country Code"): a 8-bit long value that codes the country where the server provides service. These country codes should be known to all international systems where federation is possible.

In addition, the server maintains a local database, IDTable.

[BCS-01] The Back-end set up shall be done according to the procedure in clause 6.2.1.

6.2.2 Application Registration (Server side) and IDTable database

For each registered application App_A, belonging to an anonymous user U_A, the server Srv keeps a record in a local database IDTable with the following information:

- K_A^{auth} ("Authentication Key for App_A"): an L-bit long key, with $L \geq 128$ to be defined, shared with App_A, that is used to authenticate App_A messages.
- K_A^{enc} ("Encryption Key for App_A"): a L-bit long key, with $L \geq 128$ to be defined, shared with App_A, that is used to encrypt sensitive information sent from the server to App_A.
- ID_A ("Permanent IDentifier for App_A"): a 40-bit identifier that is unique for each registered App, and generated randomly (random drawing process without replacement to avoid collisions). This permanent ID is only known to the server.

- UN_A ("User U_A Notified"): this flag indicates if the associated user has already been notified to be at risk of exposure (*true*) or not (*false*). It is initialized with value *false*. Once set to *true*, the user is not allowed to make any additional status request. The flag can be reset if the user can prove that he/she is not at risk anymore (for example by proving that he/she got a test and the result was negative).

- SRE_A ("Status Request Epoch"): a 24-bit value that indicates the last epoch when U_A has sent a "Status Request".

- LEE_A ("List of Exposed Epoch(s)"): each time one of U_A's EBIDs appears in the proximity list of an diagnosed user, the epoch j when the encounter happened between the diagnosed user and U_A is added to this list. This list of epochs reflects the exposure of the user (temporal and frequency information) and is used, together with other information (e.g. the duration and proximity of a contact to an diagnosed person), to compute the risk score. Note that a given epoch i can appear several times in this list if several HELLO messages sent by App_A appeared in the proximity list(s) of one of several diagnosed users during that epoch.

[BCR-01] Registration on the Back-end shall be done according to the procedure in clause 6.2.2.

6.2.3 Application Registration (Application Side)

A user U_A who wants to install the application on his/her device downloads it from the relevant digital distribution platform. After installing the application App_A, U_A needs to register to the back-end server. During this registration phase:

- A proof-of-work system (or a system which is able to differentiate between humans and machines) is implemented in order to avoid automatic registrations by bots or denial of service attacks.

- App_A, is configured (over a TLS channel) with:

  - The current epoch value, i.

  - The duration of an epoch, *epoch_duration_sec*.

  - The starting time of the following epoch, $T_{ptsstart} + (i + 1) * epoch\ duration\ sec$ (this is required for synchronization with the server).

  - The keys K_A^{auth} and K_A^{enc}, shared with the server.

  - An initial list of T ($EBID_{A,i}$,$ECC_{A,i}$) pairs.

The keys K_A^{auth} and K_A^{enc} shall exchange by means of a secure key establishment procedure an example could be found here [i.3].

[BCR-02] Registration on the Mobile Application shall be done according to the procedure in clause 6.2.3.

6.2.4 Generation of the Ephemeral Bluetooth® Identifiers

During registration and then regularly, i.e. every M epochs (value to be defined), each registered user U_A connects to the server in order to obtain a list of the M ($EBID_{A,i}$, $ECC_{A,i}$) pairs for the M following epochs. For efficiency, this request can be performed together with an Exposure Status Request.

Upon receiving such a request, the server generates and sends to App_A an encrypted list of T ($EBID_{A,i}$, $ECC_{A,i}$) pairs for the upcoming T epochs, where:

- $EBID_{A,i}$ ("Ephemeral Bluetooth® IDentifier for A"): a 64-bit identifier generated for the epoch i as:

  - $EBID_{A,i} = \mathrm{ENC}(K_S, i \mid ID_A)$

    where ENC is a 64-bit block cipher, for example, SKINNY −64/192 (a block cipher of block size 64 bits and key size 192 bits) (ETSI GS E4P 006 [2]).

- $ECCA_{,i}$ ("Encrypted Country Code"): an 8-bit code that indicates, in an encrypted form, the country code of $EBIDA_{,i}$. This field is used for federation purposes and can only be decrypted by back-end servers that have federation agreements. More specifically. $ECCA_{,i}$ is the country code CCA encrypted using AES-OFB with key KG and IV $EBIDA_{,i}$,: $ECCA_{,i}$ =MSB(AES(KG,$EBIDA_{,i}$ $|0^{64}$)) $\oplus$ CCA
  The encryption of the list is performed with the authenticated encryption algorithm AES−GCM, using key K_A^{enc} with a random 96-bit IV and a 128-bit tag.

[BCE-01] Ephemeral identifiers generation shall be done according to the procedure in clause 6.2.4.

6.2.5 Diagnosed User Declaration

Upload Authorization Procedure:

If user U$_C$ is tested and diagnosed and it is estimated that he/she was contagious from $ContStart_C$ to $ContEnd_C$ (expressed in seconds using the standard NTP "Seconds" system(IETF RFC 5905 [7])), he/she is proposed to upload to the server each (HELLO,*Time*) pair of his/her *LocalProximityList* that satisfies:

$$\text{ContStart}_C < \text{Time} < \text{ContEnd}_C$$

The present document does not detail the interactions between $AppC$ and the Health Authority. In particular it does not present the security/authorization procedure that verifies that only authorized and positively-tested users are allowed to upload their *LocalProximityList.*

NOTE 1: During this upload, $AppC$ does not reveal any of its EBIDs to the server.

Upload Mechanism:

A *LocalProximityList* contains the EBIDs of the devices that the diagnosed user has encountered in the last *CT* days. For privacy purposes, it is necessary to "break" the link between any two EBIDs contained in the *LocalProximityList* to prevent the server from getting this information. Therefore, instead of uploading the *LocalProximityList* this scheme uploads each of its elements independently.

Different solutions can be envisioned to achieve this goal: by using a Mixnet, or by uploading to a trusted server (for example to a hospital or to a health organization) that mixes all the (HELLO,*Time*) pairs of all diagnosed diagnosedusers' *LocalProximityList* or thanks to a secure hardware component that processes the uploads of the *LocalProximityList*.

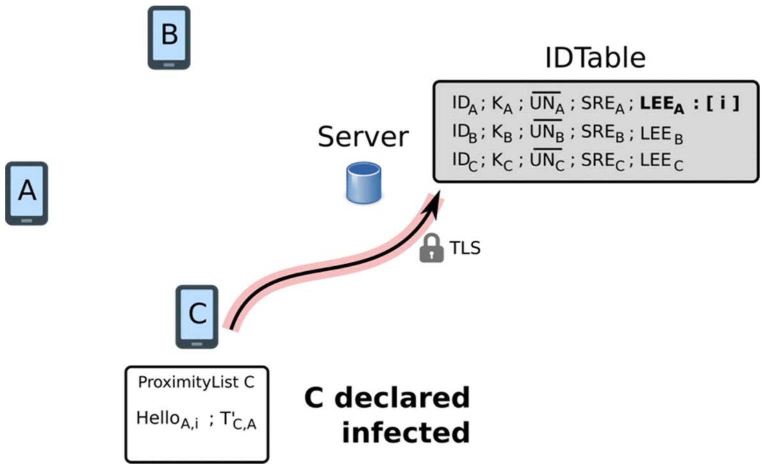
B
IDTable
ID$_A$; K$_A$; $\overline{UN}_A$; SRE$_A$; **LEE$_A$: [i]**
ID$_B$; K$_B$; $\overline{UN}_B$; SRE$_B$; LEE$_B$
ID$_C$; K$_C$; $\overline{UN}_C$; SRE$_C$; LEE$_C$
Server
A
TLS
C
ProximityList C
Hello$_{A,i}$; T'$_{C,A}$
C declared infected

Figure 6: Diagnosed Node Declaration: upon agreement, a user tested positive uploads his/her *LocalProximityList* to the server

Server Operations:

Upon reception of a h_A = ($HELLO_A$,$Time_A$), the server:

1) parses h_A to retrieve ecc_A (8 bits), $ebid_A$ (64 bits), t_A (16 bits), mac_A (40 bits) and $Time_A$ (32 bits);

2) checks that:

   $|t_A - \mathrm{TRUNC}_{16}(Time_A)| < \delta$
   where δ is a configurable time tolerance (typically a few seconds, value to be defined). h_A is rejected if this test is not satisfied;

3) decrypts ecc_A, using K_G, to recover the message country code, cc_A. If cc_A is different from the server's country code, CC_S, and corresponds to a valid country code, h_A is forwarded to the back-end server managing this country using the federation procedure in place;

4) computes $\mathrm{ENC}^{-1}(K_S, ebid_A)$ to retrieve $i_A \mid id_A$, where i_A is the epoch number corresponding to $ebid_A$ and id_A is the permanent identifier;

5) verifies that idA corresponds to the ID of a registered user, otherwise hA is rejected silently;

6) checks that TimeA corresponds to epoch iA:

   $|(Time_A - T_{tpsstart})/epoch\_duration\_sec - i_A| \leq 1$

NOTE 2: A difference of 1 may happen if the HELLO message is sent at the very end of the epoch due to transmission and processing times). h_A is rejected if this test is not satisfied.

7) retrieves from IDTable, K_A, the key associated with id_A;

8) verifies if the MAC, mac_A, is valid as follows: mac_A ==HMAC−SHA256($KA, c1 \mid ecc_A \mid ebid_A \mid t_A$). If mac_A is invalid, h_A is rejected silently;

9) adds i_A in LEE_A;

10) erases ($HELLO_A$, $Time_A$).

[BCD-01] The Back-end shall follow the procedure in clause 6.2.5 when a User is diagnosed.

6.2.6 Exposure Status Request (ESR)

To check whether user *UA* is "at risk", i.e. if he/she has encountered diagnosed users in the last *CT* days, application *AppA* regularly sends "Exposure Status" Requests (ESR REQUEST) to the server *Srv* for *IDA*. The server then computes a "risk score" value, derived in part from the list *LEEA* corresponding to *IDA*. The server replies with a ESR REPLY message that is set to "1" when the user is "at risk" (i.e. if the "risk score" is larger than a threshold value) or to "0" otherwise.

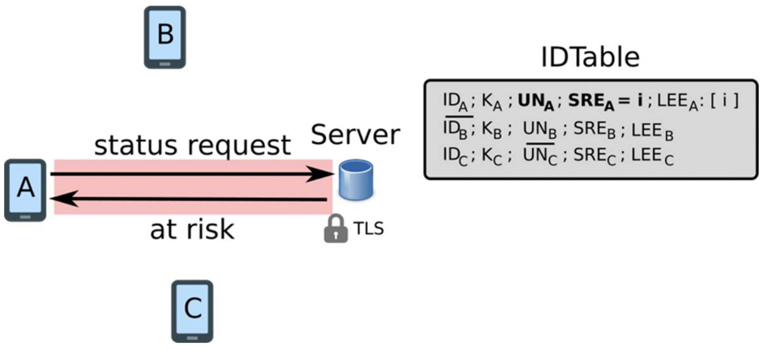
B
IDTable
ID_A ; K_A ; $\mathbf{UN_A}$; $\mathbf{SRE_A = i}$; LEE_A: [i]
$\overline{ID_B}$; K_B ; $\underline{UN_B}$; SRE_B ; LEE_B
ID_C ; K_C ; $\overline{UN_C}$; SRE_C ; LEE_C
status request
Server
A
at risk
TLS
C

Figure 7: Exposure Status Request: The App regularly requests the server to know if any of its EBIDs appeared in any HELLO messages collected by an diagnosed user

Application processing:

AppA queries the server by sending the following request over an TLS channel:

$$\mathrm{ESR\_REQUEST}_{A,i} = [\mathrm{EBID}_{A,i} \mid i \mid \mathrm{Time} \mid \mathrm{MAC}_A]$$

with: $\mathrm{MAC}_{A,i} = \mathrm{HMAC}-\mathrm{SHA256}(K_A^{auth}, c2 \mid \mathrm{EBIDA}_{A,i} \mid i \mid Time)$

where:

- $c2$: Fixed prefix "02" (8 bits).
- $EBID_{A,i}$: Ephemeral Bluetooth® ID of the epoch i (64 bits).
- i: epoch of validity of $EBID_{A,i}$, either the current epoch if App_A has an EBID for the current epoch or the latest epoch for which App_A has an EBID (24 bits).
- *Time*: 32-bit timestamp in seconds, corresponding to the transmission time.

[BCX-01] The Mobile Application shall follow the procedure in clause 6.2.6 to check the status of its User.

Server processing:

Upon the reception of a request, ESR_REQUEST$_{\mathrm{A,i}}$, at epoch i, the server:

1) parses ESR_REQUEST$_{\mathrm{A,i}}$ to retrieve $ebid_A$, i_A, $time_A$ and mac_A;

2) verifies that $time_A$ is close to its current time;

3) retrieves the permanent identifier id_A and epoch i'_A by decrypting $ebid_A$, as : $i'_A \mid id_A = \mathrm{ENC}^{-1}(K_S, ebid_A)$;

4) verifies that $i_A == i'_A$, otherwise the message is rejected;

5) uses id_A to retrieve from IDTable the associated entries: K_A^{auth}, UN_A, SRE_A, LEE_A;

6) verifies that: $mac_{A,i}$ ==HMAC−SHA256(K_A^{auth}, $c2 \mid ebid_A \mid i \mid t$). If $mac_{A,i}$ is incorrect, the message is silently rejected;

7) verifies that UN_A is "false" (User Notified) in order to check that the user has not already received a "At Risk" notification. If the user has already been notified, the message is silently rejected;

8) verifies that $(i - SRE_A) \geq T$, where T is minimum number of epochs between two consecutive ESR+REQUEST. Otherwise, the message is silently rejected.

If the $ESR{+}REQUEST_{A,i}$ is valid, the server:

1) updates $SREA$ with the current epoch number, i, in IDTable.

2) computes a "risk score" value, derived in part from the list $LEEA$.

3) Two situations are then possible:

   - If the computed score indicates that the user is at risk of exposure, the server sets UN_A at "true". It means that App_A cannot perform any new request until user U_A is tested and his/her status updated. An ESR REPLY$_{\mathrm{A,i}}$ message set to "1" (at risk of exposure) is then returned to the user.

   - If the computed score does not indicate any significant risk, an ESR_REPLY$_{\mathrm{A,i}}$ message set to "0" is returned to the user.

[BCX-02] The Back-end shall follow the procedure in clause 6.2.6 when asked to check the status of a User.

Application processing:

Upon receiving the $ESR\_REPLY_{A,i}$ message:

1) App_A replies with an ACK message.

2) If $ESR\_REPLY_{A,i}$ is set to "1":

   - App_A keeps broadcasting HELLO messages but stops sending ESR REQUEST requests to the server.

- *UA*, receives a notification from *AppA* with some instructions (for example to go to the hospital to get tested, to call a specific number or go into quarantine).

[BCX-03] The Back-end shall follow the procedure in clause 6.2.6 to process the answer of the Back-end to check the status of its User.

7 Privacy, data and system security

7.1 Risk analysis

This clause provides a risk analysis with respect to the basic design principles of the centralized and the decentralized approaches to DCTSs.

Centralized design approaches for proximity tracing conduct all critical operations at the back-end including the generation of ephemeral ids and the determination of individuals at risk of having been exposed to the infection. This puts significant importance on the responsibility and security of the back-end as its breach or compromise could lead to loss of privacy of individuals. However, in contrast to the decentralized design approaches, no information about the health status of users (tested positive or signalled "at risk") is accessible to participating mobile devices: this sensitive information is either kept in the Mobile Device itself (tested positive status) or kept within the back-end ("exposed contacts" and "at risk" status).

Decentralized design approaches for proximity tracing aim to constrain the role of the back-end and limit functionalities that could be potentially abused for surveillance of participating individuals. The architecture follows design principles with the aim of using the back-end primarily for aggregating and disseminating the diagnosis keys of users tested positive to the participating Mobile Devices. The back-end does not process that data and acts solely as a communication platform. However, it is critical for the back-end to be functional for the decentralised DCTS to remain operational. The back-end collects and processes pseudonymised personal data.

It is worth noting that all digital contact tracing systems run the risk of exposing the health status of an individual when their contacts are notified.

Any DCTS (centralised or decentralised system) produces lists of contacts to break infection chains, those lists of contacts shall not be used for any other purpose.

Centralised contact information collection might enable data enrichment and epidemiological benefits that include, for example, earlier R-number calculation, the spotting of variants by R-number changes, spotting asymptomatic carriers, and facilitating second and third order notifications rather than only first order notifications. A centralised DCTS does introduce the risk that such data could be misused or linked to other data sources, allowing individuals to be identified for law enforcement or other non-medical tracing activities. Technical, organizational, and legal measures can be used to lower the risk such as introducing 'fake' contact IDs to dilute the ability for a real contact ID's owner to be identified. As a policy measure, strong data isolation and access regimes, including strong legislation to protect privacy for contact tracing specific data, could alleviate this risk too.

A decentralised DCTS has the privacy benefit of the contact information not being disclosed to a public health authority. This comes at a cost of epidemiological information enrichment to slow the pace of an outbreak. The decentralised DCTS is not without its own risks of user identification. By collecting IDs today, and perhaps recording video or photos of those visiting a location, and then downloading the IDs of infected people later, it would be possible to identify those who have become ill. Additionally, technical measures should be implemented to lower the risk of relay or replay attacks which otherwise can lead to false exposure notifications. Measures for this include cryptographical means as wells as an out-of-band illness confirmation mechanism such as entering a test result token which can then be used to verify the user's health status. This might allow the linking of pseudonymous ephemeral IDs to the test, and thus identity, of the tested person by the public health authority. Again, technical, organizational, and legal measures of strong data isolation and access regimes, and potentially strong legislation to protect privacy would still be needed.

The choice between systems comes down to where effective control of the risks can be made, and of an individual society's trust in any system's operator. Technical, organizational, and legal measures can be put in place in both types of systems to alleviate concerns, but it will not remove those concerns. An oversight regime might be easier to audit and have its proper use validated in a single centralised system rather than detecting a large number of cases of misuse by individual actors across an entire country. Conversely though, a centralised system is a tempting target for citizens' own operating governments to misuse. It therefore falls to each state and population to decide for themselves how to operate and trust any DCTS within the norms and expectations of their own societies. The following table summarizes some of the main possible vulnerabilities of the centralized and the decentralized approach [i.8].

Table 1: Centralized exposed-keys (e.g. ROBERT)

Risk	Vulnerability to risk
Tracking of users	In ROBERT a central authority anonymously registers users, assigning to each of these users a long-term pseudonym. The ephemeral identifiers used by each user are mapped to this long-term pseudonym by a trapdoor function (symmetric encryption algorithm). Anyone who obtains the trapdoor (i.e. gets access to the symmetric key used e.g. an external attacker or a corrupted authority) would be able to link all observed or collected ephemeral identifiers generated with the help of that trapdoor (symmetric key) to the corresponding pseudonym of the user. This would allow the potential tracking of users. A corrupted authority could also map the user identity to the long-term pseudonym during the registration or renewal procedure.
Building contact graph	In ROBERT, users who report an infection, reveal to the server all the ephemeral ids of the contacts they had during the last days. This information is used to evaluate the risk of people who were in contact with the user reporting the infection, but it is not stored in the server. A corrupt authority or an attacker that has compromised the central server or the communication network could build a contact graph for the users reporting infection and the persons they have contacted (either a graph of the ephemeral ids of the users or in the worst case of the identities of the users).
Identification of diagnosed users	The attack is possible if a user registers a dummy account and an app which is used only when in contact with a specific user. If this app is notified as at-risk, the attacker can know that the specific user was diagnosed. ROBERT is not vulnerable to this attack at large scale.

Table 2: Decentralized diagnoses-keys (e.g. DP3T/GAEN)

Risk	Vulnerability to risk
Tracking of users	In DP3T/GAEN ephemeral identifiers a generated from a medium-term key which is changed periodically (e.g. daily). When a user reports an infection, these diagnosis keys are broadcasted. Once these diagnosis keys are known, it may be possible to track this user, meaning that it may be possible to identify the user. This attack is relatively easy to do (as diagnosis keys are broadcasted) but reveals small amounts of information as only concerns to diagnosed users from which the attacker has recorded ephemeral identifiers at given locations. The low-cost design allows local tracking of users during the past window where the ephemeral identifiers are linkable. More advanced attackers could also modify the app to record much more information than intended by the protocol, this could be mitigated by running the proximity tracing app within a trusted execution environment (TEE).
Building contact graph	In DP3T/GAEN a user A can obtain a proof of a contact with another user B by storing ephemeral ids in a blockchain commitment of contacts and comparing this values generated from diagnosis keys reported by the user B. This could, for instance, used for blackmail. A compromised back-end could also know whether A was in contact with B by injecting ephemeral identifiers of A to create a false notification and observing whether B is notified (e.g. B goes to self-isolation). This attack requires not only compromising the back-end, but also register ephemeral ids of A and have side information about B.
Identification of diagnosed users	The main vulnerability of DP3T/GAEN lies in that the ephemeral ids of diagnosed users are publicly revealed. This implies that is relatively easy to identify users that have reported infection. Several versions of this attack are possible (Paparazzi, attack, Nerd attack, Militia attack, etc.). A resourceful adversary could potentially collect a large number of ephemeral identifiers using powerful antennas and then compare them with the list of diagnosed ephemeral identifiers to track the movement of diagnosed users in a small area for the low-cost version of the protocol, thus isolating potential disease hotspots, the unlinkable design makes this more challenging. To prevent an adversary from collecting large number of ephemeral identifiers a ***k*-out-of-*n*** secret sharing scheme is proposed where the ephemeral identifiers are split into $\boldsymbol{n}$ chunks and at least $\boldsymbol{k}$ of them are needed to reconstruct an ephemeral identifier fully, this could introduce an additional computational cost but would not limit a powerful adversary. Linking ephemeral identifiers with timing information can be used to narrow down the list of diagnosed individuals, collation with other side-channels is also possible. Traffic masking is proposed to hide the communication with the back-end by using "dummy" traffic and batching. Further anonymity is proposed by precluding the back-end from logging IP addresses. It is also suggested that the Bluetooth® Medium Access Control (MAC) address of the phone changes with ephemeral identifiers to prevent prolonged tracking. All decentralized proximity tracing systems suffer from exposing the identity of the person who might have exposed a user.

7.2 Privacy

7.2.1 Centralized approach

7.2.1.1 The case of a naïve centralized approach

With a naïve centralized approach, safeguarding the privacy of individuals is a lot harder and in stark contrast to the decentralized design as it depends upon the back-end's behaviour. The back-end server has to be trusted not to go rogue and mount active or passive attacks as this would compromise the privacy. Specifically, if the authority in charge of the back-end does not introduce any privacy protection, the back-end potentially can collate personal information related to a user such as the list of IP addresses used, app installation details and other metadata with the permanent identifier and the issued ephemeral ids. The back-end potentially gets a view of the so called proximity graph (other people the user has been in contact with, according to the Proximity Detection Method) when a diagnosed user uploads their data. This information could be merged with similar proximity graphs if one of their contacts gets diagnosed.

7.2.1.2 The case of the ROBERT centralized approach and its deployment

In order to avoid the privacy issues described above, several safeguards are required, from the protocol design viewpoint, from the back-end design viewpoint, and from the legislative viewpoint.

The ROBERT protocol [i.4] requires that only the exposed contacts are uploaded to the back-end in case of a user tested positive, without any identification of the diagnosed user, and that this list be mixed with other uploaded lists to prevent leaking social graphs (a mixnet can be used for this purpose in the server sanitization frontend). The back-end design shall also include a sanitization frontend that gets rid of IP addresses and other metadata that could allow device re-identification. Similarly, raw information (like the record of a contact with a diagnosed user, namely the ephemeral ids of the contact, signal strength and timing information), once processed, shall be destroyed. The use of these privacy-by-design techniques, at the protocol and server design levels, reduces, or even avoids most of the privacy risks. Nevertheless, the pseudonymised contact list stored in a centrally controlled server, if compromised, might still be correlated with other data (e.g. publicly available data crawled from online social networks) to mount re-identification attacks as shown for other pseudonymised social graphs (see [i.9] for an overview of such attacks).

[BCP-01] The back-end shall be designed for resiliency, for instance with isolated components, potentially including a Hardware Security Module (HSM) for managing cryptographic operations, and where information is systematically destroyed as soon as it is processed, keeping only the minimum vital data.

[BCP-02] The High Level Requirement **[HL-SE-19]** from ETSI GS E4P 003 [1] shall be implemented.

However, there is no way for a User of the DCTS to verify on its own if data has indeed been deleted.

7.2.2 Decentralized approach

In the decentralized design the back-end is not responsible for protecting the privacy of individuals and is not entrusted to do so. The decentralized protocol has been designed to resist surveillance at this level and keep the privacy intact even in the event of back-end breach or gets compromised. The privacy of the participating individuals is independent of the back end's behaviour. The decentralized design does not expose any interaction information or proximity tracing graph to the backend. The ephemeral ids are independent of the user data and generated locally on the device which prevents linkage to personal information by the backend server. Nevertheless, "dummy" traffic and anonymous communication should be utilized to further reduce the privacy risks.

However, the decentralized design exposes the "diagnosed status" of users: a decentralised protocol like GAEN gathers the diagnosis keys in a database [i.4], and these diagnosis keys enable derivation of all the ephemeral ids during the contagious period of a user who has been diagnosed as positive. This database contains privacy sensitive data that can be de-anonymized via Bluetooth® LE scanning [i.5], using either a dedicated Web service [i.6] or application [i.7]. This weakness is the result of a design choice to protect the proximity graph over user infection status, although this might fall into the sensitive personal data category under the GDPR. A centralised protocol like ROBERT [i.4] is less vulnerable to these privacy threats since it never collects any such information (it remains on the user's smartphone).

Another consideration is the trust model that a decentralized approach requires with respect to storage of the collected ephemeral identifiers. In the case of GAEN, this sensitive information is collected, stored and processed by software from private companies, in components that are part of the respective operating system. These components are closed-source and cannot be audited (as of now).

Finally, it is also important to consider privacy implications while boot-strapping the contact tracing system as in the event of a low number of diagnostic keys risk of re-identification attacks might be higher. Fake diagnostic keys (dummy keys) [i.1] generated by the back-end have been used to lower re-dentification risks. The usage of dummy keys generated by the back-end has also been proposed to strengthen unlinkability when the number of users uploading diagnostic keys is too low [i.2]. This reduces the delay which could alternatively be introduced to ensure that at least a certain number of keys are published together for anonymity reasons.

One of the guiding principles of the decentralised approach was to minimize the risk of the DCTS to be used for surveillance like some competing designs readily allow. Hence, the design choices try to limit the influence of the back-end and information it collects. It has limited impact on privacy. Nevertheless, even in case of the decentralised design personal data is handled by the back-end.

[BDP-01] Therefore, the back-end shall be designed and operated in a way which complies with the relevant data protection laws, e.g. in case of Europe it shall comply with the GDPR.

7.3 Data and system security

7.3.0 Introduction

Clauses 7.3.1 and 7.3.2 cover the security properties of the back-end in the proposed centralized and decentralized approaches for proximity tracing and how each design choice handles the user data and system functionality based on key security metrics, namely, **availability**, **confidentiality**, **integrity**. **Privacy**, has been covered in clause 7.2 above.

7.3.1 Centralized approach

Availability: it is a fundamental requirement of the centralized design that the back-end remains available at all times. Since, all activities including initial setup, data collection, upload authorization, data processing and exposure status notification are controlled by the back-end, without its availability the protocol cannot function. However, devices are not supposed to be continuously connected to the Internet, the exchange of ephemeral ids between devices being autonomous.

Confidentiality: secure storage of data and especially symmetric keys at the back-end is of key importance because it processes personally identifiable information pertaining to individuals. Other sensitive data including key generation materials also reside on the server. The communication with the devices needs to be confidential. In some proposals the back-end identifies the individual at risk of exposure for notification by design and hence they lose the opportunity of voluntary reporting. Given such critical confidentiality requirements, it may be wise to consider using hardware security modules for safeguarding the key material. A distributed storage model could also be useful in de-aggregating the generated ephemeral ids during storage and protecting them from being linked.

Integrity: the generation of ephemeral ids and issuing exposure notification is handled by the back-end. These are critical parts of the proximity tracing protocol which rely on the correct implementation in the back-end. Deliberate manipulation of such data would compromise its usefulness and trust. The back-end is also responsible for validating the self-reported diagnosed status through some authorization mechanism in conjunction with the health authority. This process could be much simpler as compared to the decentralized approach as all identifiable information is generated and handled by the back-end. The information being sent to the participating devices should be authenticated to prevent abuse.

7.3.2 Decentralized approach

Availability: it is a fundamental requirement of the decentralized design that the back-end remains available at all times. The participating Mobile Devices should be able to regularly query the back-end to download the anonymized list of diagnosis keys to compute their exposure. Additionally, diagnosed users should be able to upload their diagnosis keys. Hence, the availability of the back-end is imperative for the proximity tracing protocol to function.

Confidentiality: the back-end is responsible for secure storage of information when authorized users upload their diagnosis keys. It shall also support secure communication to protect against eavesdroppers and active attackers. The communication with Mobile Devices should also be masked (e.g. using "dummy" traffic and anonymous communication, e.g. mixnet) to prevent traffic analysis attacks to discern the identity of the diagnosed individuals. This requires participation of all devices in sending "dummy" traffic to the back-end. No information linking the identity of the diagnosed person (collected through voluntary exposure upload) should be stored by the back-end. It is therefore useful to utilize means for anonymous communication.

Integrity: the back-end is trusted to not manipulate the data it receives or transmits. It should not introduce fake exposure events or remove reported exposure events. This does not impact the confidentiality or privacy of the system, but it is important for the protocol to work as intended. The communication between the back-end and the Mobile Devices shall be secured to protect it from manipulations by third parties and eavesdroppers. The back-end is also tasked with authorizing the diagnosis keys upload by the Mobile Devices. In the event of an exposure the diagnosed individuals are provided with relevant authorization information by the local public health authority. This information can be used by the diagnosed individuals to authorize themselves and voluntarily upload their diagnosis keys to the back-end. It is the responsibility of the back-end to verify that the diagnosis keys are uploaded by an authorized user.

Annex A (informative): Mapping of Back-end Features to Requirements and Clauses

Requirements are identified by three letters and subsequent consecutive numbers when they belong to one Back-end feature. The following list provides a mapping table for easy access and resolution of acronyms.

Backend Feature	Requirement number	Described in clause
Back-end Decentralized Data Structures	[BDD-01]	6.1.1
Back-end Decentralized Transport Encryption	[BDT-01] - [BDT-02]	6.1.2
Back-end Decentralized Key Download	[BDK-01]	6.1.3
Back-end Decentralized Base URL	[BDB-01] - [BDB-2]	6.1.3
Backend Decentralized Key Upload	[BDU-01] - [BDU-4]	6.1.4
Back-end Decentralized Test Results	[BDR-01] - [BDR-6]	6.1.5 and 6.1.6
Back-end Decentralized One-Time Token	[BDO-01] - [BDO-14]	6.1.7
Back-end Decentralized Configuration Parameters	[BDP-01]	6.1.8
Back-end Centralized Server Setup	[BCS-01]	6.2.1
Back-end Centralized Registration	[BCR-01] - [BCR-2]	6.2.2 and 6.2.3
Back-end Centralized Ephemeral Bluetooth® IDs	[BCE-01]	6.2.4
Back-end Centralized Diagnosed User	[BCD-01]	6.2.5
Back-end Centralized Exposure Status Request	[BCX-01] - [BCX-3]	6.2.6
Back-end Centralized Privacy	[BCP-01] - [BCP-2]	7.2.1.2
Back-end Decentralized Privacy	[BDP-01]	7.2.2

Annex B (informative): Matching with ETSI GS E4P 003 'Requirements for Pandemic Contact Tracing Systems using mobile devices'

Device requirement	**System requirement** **(from ETSI GS E4P 003 [1] 'Requirement for Pandemic Contact Tracing Systems using mobile devices')**	**Decentralized approach** **Applicable to GAEN**	**Centralized approach** **Applicable to DESIRE**
[BDD-01]	[HL-PV-11]	Yes	
[BDT-01] - [BDT-02]	[HL-SE-07]	Yes	
[BDK-01]	[HL-PV-11] [HL-PV-12]	Yes	
[BDB-01] - [BDB-2]		Yes	
[BDU-01] - [BDU-4]	[HL-SE-10] [HL-PV-11] [HL-PV-12]	Yes	
[BDR-01] - [BDR-6]	[HL-PV-11] [HL-PV-12]	Yes	
[BDO-01] - [BDO-14]	[HL-PV-12]	Yes	
[BDP-01]	[HL-GE-07]	Yes	
[BCS-01]	[HL-SE-07] [HL-PV-07]		Yes
[BCR-01] - [BCR-2]	[HL-SE-07] [HL-PV-07]		Yes
[BCE-01]	[HL-PV-06] [HL-PV-07]		Yes
[BCD-01]	[HL-PV-11] [HL-PV-12]		Yes
[BCX-01] - [BCX-3]	[HL-PV-11] [HL-PV-12]		Yes
[BCP-01] - [BCP-2]	[HL-SE-19]		Yes
[BDP-01]	[HL-GE-01]	Yes	

History

Document history		
V1.1.1	May 2021	Publication