The Cluster Exposure Verification (CLEA) Protocol: Specifications of Protocol Version 0

Vincent Roca, Antoine Boutet, Claude Castelluccia

Contents

The Cluster Exposure Verification (CLEA) Protocol: Specifications of Protocol Version 0

Vincent Roca, Antoine Boutet, Claude Castelluccia

PRIVATICS team, Inria, France

{firstname.lastname}@inria.fr

June 8th, 2021

This document and the associated reference source code are available at: [https://gitlab.inria.fr/stopcovid19/CLEA-exposure-verification]
and mirrored at: [https://github.com/TousAntiCovid/CLEA-exposure-verification]
This document is also available in the HAL open-archive: [https://hal.inria.fr/hal-03146022]

1- Introduction

This document is a specification of the CLuster Exposure verificAtion (CLEA) protocol meant to warn the participants of a private event (e.g., wedding or private party) or the persons present in a commercial or public location (e.g., bar, restaurant, sport center, or train), when this event or location later became a cluster because a certain number of people who were present at the same time have been tested COVID+.

It is based:

- on a central server, in order to automatically detect potential clusters. This server is under the responsibility of a health authority;
- on the display a QR code at the location or on a ticket, either in a static (e.g., printed) or dynamic manner (e.g., via a dedicated device, smartphone, or tablet);
- and on a smartphone application.

This smartphone application is used to scan a QR code, to store it locally for the next 14 days, and to perform periodic risk analyses, in a decentralized manner, on the smartphone. In order to enable this decentralized risk analysis, information about clusters (i.e., the location pseudonyms and timing information) needs to be disclosed. We believe this is an acceptable downside because this information is not per se sensitive health data (it does not reveal any user health information to an eavesdropper), although it can be considered as personal data (it is indirectly linked to the location manager).

This specification only considers the synchronous scan of a QR code, for situations where a user scans a QR code upon entering an event or location (e.g., a restaurant).

Finally, the CLEA protocol is also meant to be used by the location employees in order to warn them if their work place is qualified as cluster, or on the opposite to let them upload information to the server if they are themselves tested COVID+.

2- CLEA protocol high level principles

2.1- Terminology

The following terms are used in this document:

Name	Description
`MCT`	Manual Contact Tracing team.
Location	Synonymous to venue, this is a closed area where people meet. It can be a private location (e.g., for a wedding or a party event) or a commercial or public location (e.g., a bar, a restaurant, a sport center, an entertainment hall or auditorium, a train).
Device	A specialized device, or a general purpose smartphone or tablet with the appropriate software, used by the location manager or event organizer, that displays a QR code. Alternatively, QR codes can also be printed or included in a digital ticket.
(User) terminal	The user smartphone.
CLEA application	The application on the user terminal (smartphone).
QR code	The dynamic or static QR code of a location that is scanned (e.g., when entering the location). It contains a URL ("deep link") structured as: `"country-specific-prefix" "Base64url(location-specific-part)"`.
Location Specific Part	This is the part of the QR code that contains the information specific to the location. With a dynamic QR code, the information contained in this part is periodically renewed.
Period	With dynamic QR codes, time is split into periods (e.g., 24 hours), during which the location pseudonyms (more precisely a temporary cryptographic key and a derived temporary UUID) are stable. After that period, a new location pseudonym is generated. For practical reasons, a new period MUST start at a round predefined hour (e.g., 4:00am may be chosen as a default period start). As a special case, and this is used by static QR codes, a period can also have an unlimited duration, meaning that the location pseudonym will remain unchanged.

2.2- Overview

Two key design choices: centralized cluster detection and decentralized risk estimation and notification
This protocol must comply with two privacy-related requirements:

- the location manager must not be able to collect any data with respect to the clients (unlike written records where clients fill-in their name and contact information), and
- the amount of information uploaded by the CLEA application to the central server must be minimized.

In practice, no information is uploaded to the server unless a client is tested COVID+. In that case, if the user explicitly agrees (informed consent), the application uploads the list of scanned QR codes during the past 14 days[1] along with timing information to the central server, in order to enable a ***centralized anonymous cluster***

[1]The 14 days number is provided as an example. The national health authority will define the appropriate epidemiological value that is considered the most appropriate, that may also depend on another considerations like the date of first symptoms when known. The

detection. The server can detect clusters by considering the number of COVID+ users in a location at the same time, without having access to the name nor address of this location. Then this central server updates its list of location temporary pseudonyms and time (with an hour granularity by default) corresponding to clusters.

Figure 1: Centralized cluster detection. Here Alice, tested COVID+, agrees to upload her scanned QR codes to the CLEA backend server, which, after verifying the validity of the upload, identifies if some of the visited locations needs to be qualified as potential cluster.

In parallel, each CLEA application periodically downloads this list containing the latest clusters that have been identified, in order to check locally whether or not there is a match. In case of a match, the user is informed with a "warning". The exact type of warning message could be adjusted to reflect the risk level (e.g., if a high number of COVID+ users have been identified in a cluster), which is out of scope of the present specification. Therefore CLEA performs a ***decentralized risk evaluation***.

Figure 2: Decentralized risk evaluation. Here Bob compares his scanned QR codes with the new potential cluster location pseudonyms in a first step, and if a match is found, if the corresponding period overlaps significantly with his own presence as stored in his local database.

We believe that making public the list of location temporary UUIDs and time corresponding to clusters is an acceptable tradeoff, because this information is not per se sensitive health data (it does not reveal any user health information to an eavesdropper), although it can be considered as personal data (it is associated to the location manager)[2].

A single protocol, three potential deployments Central to the deployment of CLEA is the question of the role given to the Manual Contact Tracing Team (MCT). Three options exist:

- Option 1: the MCT is at the center, for a maximum control. Here the upload of Alice scanned QR code history is done during or after an interview with the MCT, under MCT control;

Figure 3: CLEA deployment option 1, with the MCT at the center.

- Option 2: the MCT is at the edge, for maximum scalability and speed, and to avoid overloading the MCT. Here clusters are identified as soon as Alice uploads her scanned QR code history, without any delay. The MCT is also informed of those new clusters, yet they are not in the critical path;

Figure 4: CLEA deployment option 2, with the MCT at the edge.

- Option 3: the MCT is not involved in any manner. As a direct consequence, it is not possible to couple the digital system with any hand-written attendance register.

Figure 5: CLEA deployment option 3, involving no MCT.

Choosing an option is a local decision, based on local criteria, that does not compromize interoperability with other types of deployment in neighbouring countries.

QR codes for synchronous scans Regardless of which deployment option is chosen, the present specification focusses on use-cases involving a **synchronous scan** of a QR code, for situations where a user scans a QR code upon entering an event or location[3]. This is typically the case with a restaurant. The QR code requires a synchronous scan (i.e., when entering a location), and the location check-in timestamp is the scanning time.

details are out of scope of this document.

[2]This is a big difference with a decentralized contact tracing system, for instance based on the Google/Apple Exposure Notification (GAEN) component, where the pseudonyms of COVID+ users are freely available over the Internet: revealing sensitive health data enables any curious neighbour who uses a dedicated BLE scanning system (and https://coronadetective.eu has shown how trivial this can be since a web browser is sufficient) to immediately identify the health status of their neighbours if they upload their pseudonyms later on, with potentially major discrimination consequences. With CLEA, a decentralized risk evaluation approach makes sense as it does not disclose sensitive health information.

[3]A future version of the present specification may consider additional use-cases involving an asynchronous scan of a QR code, for situations where a QR code is produced that can be scanned either in advance or after visiting the event or location. This is typically the case with an on-line train ticketing service, where the QR code is printed on the ticket itself to let the user scan it at its discretion. The QR code enables an asynchronous scan, before, during, or after visiting the location, and the location check-in timestamp is the one provided in the QR code itself rather than the scanning time.

Static versus dynamic QR codes In order to further improve privacy and security, the current specification defines ***dynamic QR codes*** that are periodically renewed and displayed with the help of a device. Each QR code includes, among other things, the location temporary UUID (behaving as a temporary pseudonym) that, for instance, changes once a day. These dynamic QR codes necessarily require a synchronous scan, since the QR code will change over the time.

The current specification can also be used with ***static QR codes*** (e.g., printed on paper and made available to clients) if a location does not own a dedicated device. Being static, this solution has downsides: it is less robust in front of relay attacks, and it enables an attacker to display all the clusters on a map (since the location pseudonyms will not change, it is relatively easy to collect them all), or to focus on a specific set of locations to know if they have been cluster. When possible, a good practice is to regularly change static QR codes, manually, in particular if the location is identified as cluster. This aspect is out of scope of the present specification.

It can also be noticed that both static and dynamic QR codes are processed homogeneously by the same CLEA protocol, the same application and central server.

The particular case of employees Finally the employees of a location can benefit from the service, in order to be warned if their workplace is a cluster, or, on the opposite, to inform the server that they have been tested COVID+. Since they have a long presence in the location, the employees must scan a specific QR code which slightly differs from regular QR codes scanned by clients.

2.3- Attacker model and trust considerations

This specification considers two different types of attackers.

The first type is composed of individuals who try to corrupt the service, deny the service, or break the confidentiality of the service. Although a certain number of measures are taken to mitigate risks, for instance with dynamic QR codes, there are limits. For instance, the CLEA protocol cannot prevent a static QR code to be communicated to other persons, potentially tested COVID+, in order to trigger wrong cluster detections. This is also a direct consequence of a fully anonymous system that is meant to preserve user privacy.

In the second type, the authority that operates the CLEA system could try to know as much as possible on the users. Yet the system is expected to be audited by an external trusted Data Protection Authority (DPA, for instance CNIL in case of France). Because of these audits, the authority in charge of the CLEA system is assumed to be honest: it will not try to modify the CLEA protocol itself, nor the implementation of the CLEA protocol, since this would be detected by the DPA. However, it may benefit from the recorded information to infer additional information or use it for different purposes.

It follows that the CLEA server needs to be split into several independent entities: a frontend that collects the traffic from the CLEA users and sanitizes the traffic, removing the source IP address for instance, "on-the-fly", without storing any piece of information beyond what is strictly required (care should be put to logs for instance). On the opposite, the backend only processes messages that have been sanitized by the frontend. The backend may also leverage specific hardware for storing system keys, in order to minimize the security risks in case of intrusion.

It is assumed that the authority in charge of Manual Contact Tracing (MCT) is trustworthy when it comes to dealing with personal data (e.g., when an MCT team contacts a location manager or event organizer), so that it does not take advantage of the information collected beyond what is strictly required to perform its task. However this authority must not be involved in the cluster detection process, that is not under its responsibility.

It should be noted that detailed implementation choices (e.g., the exact design of the CLEA server or application) are out of scope of the present document, whereas such considerations could also impact security and privacy properties.

2.4- Technical requirements

Several technical requirements, in particular motivated by the use of embedded devices, have shaped the design:

- each QR code contains a country specific URL ("deep link"), composed of a country specific prefix (for instance: `https://tac.gouv.fr?v=0#` in case of France), and a location specific part, defined in Section Dynamic QR code generation within the device. Therefore, any binary information of the location specific part, is first translated to a printable character, using a Base64url encoding (i.e., an URL and filename safe variant of Base 64), which adds a 33% overhead compared to the binary size (see RFC4648 section 5.). Since the output of a

Base64url encoding uses an alphabet of 65 characters, it is not compatible with the Alphanumeric Mode of a QR code (limited to 45 printable characters), and it requires the use of the 8-bit Byte Mode (see QRcode18004, Section~8.4.4).

- the need to easily and reliably scan a QR code type 2 and the screen size/resolution constraints of the specialized device (e.g., 200 x 200 pixels) impact the maximum QR code size. This specification targets a Level 12 QR code Type 2 (see QRcodeWeb), of size 65x65. It also uses the 8-bit byte mode (in particular because the `#` character is absent from the alphanumeric mode), and the information capacity ranges between 155 and 367 binary characters, depending on the chosen redundancy. With maximum sized QR codes, the redundancy is set to the Medium level, leaving a maximum of 287 characters for the URL. When the URL is shorter (i.e., when `locContactMsg` is absent, see below), the redundancy is set to Q level for a better error correction feature, leaving a maximum of 203 characters for the URL. In both cases, the content of the location specific part, before Base64url encoding, uses a binary format (rather than JSON or Protobuf) for compactness reasons, in order to comply with the 287 or 203 character size limits.
- a specialized device is typically not connected to the Internet nor any wireless network, it does not feature any connector (no USB), and is powered by a non-rechargeable battery (no power plug, an autonomy of several months being expected). This specialized device will typically feature a dedicated micro-controller (e.g., a MICROCHIP micro-controller, PIC32MM0256GPM036-I/M2), with low computation capabilities, which also limits the QR code renewal period. These considerations have been considered in the present CLEA design.
- a dedicated tablet could easily remove some of the above limitations, but on the other hand a tablet is more costly, is subject to theft, and is subject to attacks, being connected to wireless networks. It is therefore a potential target device for displaying dynamic QR codes, but not the privileged one.

3- Detailed operational description

3.1- Acronyms

The following acronyms and variable names are used:

Short name	Full Name	Description
`LSP`	locationSpecificPart	The QR code of a location, at any moment, contains a URL ("deep link"), structured as: `"country-specific-prefix" "Base64url(location-specific-part)"`. The location specific part, renewed periodically with a dynamic QR code, contains information related to the location.
`SK_L`	permanentLocationSecretKey	Permanent location 408-bits secret key. This key is never communicated, but is shared by all the location devices. For instance, this key can be stored in a protected stable memory of a dedicated device (or set of devices) by the manufacturer. The manufacturer should also keep a record of this `SK_L` in a secure place if the location manager later asks for additional devices. An appropriate location manager authentication mechanism needs to be defined for that purpose that is out of the scope of this document.
`{PK_SA, SK_SA}`	serverAuthorityPublicKey / SecretKey	Public/secret ECDH key pair of the Authority in charge of the backend server. The public key is known by all devices.
`{PK_MCTA, SK_MCTA}`	manualCTAuthorityPublicKey / SecretKey	Public/secret ECDH key pair of the Authority in charge of the manual contact tracing. The public key is known by all devices. It is assumed that this authority is different from the authority in charge of the backend server.
`LTKey`	locationTemporarySecretKey	Location temporary 256-bits secret key, specific to a given Location at a given period. This key is never communicated outside of the device(s).

Short name	Full Name	Description
`LTId`	locationTemporaryPublicID	Location temporary public universally unique 128-bits Identifier (UUID), specific to a given location at a given period. This public location identifier is derived from the associated secret location key.
`t_periodStart` (in seconds)	periodStartingTime	Starting time of the period, expressed as the number of seconds since January 1st, 1900 (NTP timestamp limited to the 32-bit seconds field), by convention in the UTC (Coordinated Universal Time) timezone. A period necessarily starts at a round hour.
`ct_periodStart`	compressedPeriodStartingTime	Compressed form of the period starting time, obtained by dividing `t_periodStart` by 3600, which is guaranteed to be an integral value since a period necessarily starts at a round hour (i.e., is multiple of 3600).
`periodDuration` (in hours)	idem	Duration of the period, expressed as the number of hours. This duration is transmitted in a 1-byte field. Value 0 is invalid and should not be used, values between 1 and 254 inclusive indicate a period duration of 254 hours (i.e., 10 days and 14 hours), value 255 is reserved to the special case of an unlimited period duration. The default value is 24 hours (a period per day), but different values may be defined.
`qrCodeRenewal Interval` (in seconds)	idem	QR codes renewal interval. QR codes are renewed every `qrCodeRenewalInterval` seconds, a value of 0 indicating the QR code is never renewed during the period. This value is chosen by the device and communicated within the QR code as a power of 2, via the `CRIexp` field.
`CRIexp`	qrCodeRenewalIntervalExponent	Compact version of the `qrCodeRenewalInterval` as the exponent of a power of two, coded in 5 bits. When equal to `0x1F`, `qrCodeRenewalInterval` must be set to `0` (i.e. no renewal period), otherwise `qrCodeRenewalInterval` must be set to `2^^CRIexp` seconds.
`t_qrStart` (in seconds)	qrCodeValidityStartingTime	Starting time of the QR code validity timespan, expressed as the number of seconds since January 1st, 1900 (NTP timestamp limited to the 32-bit seconds field), by convention in UTC (Coordinated Universal Time) timezone.
`t_qrScan` (in seconds)	qrCodeScanTime	Timestamp when a user terminal scans a given QR code, expressed as the number of seconds since January 1st, 1900 (NTP timestamp limited to the 32-bit seconds field), by convention in UTC (Coordinated Universal Time) timezone.
`t_checkin`(in seconds)	checkinTime	The time when the user is enters the location, equal to `t_qrScan` with `LSPtype` is equal to 0.
`localList`	idem	Within the user terminal, this list contains all the `{QR code, t_checkin}` tuples collected by a user within the current 14-day window. Entries are added in this localList as the user visits new locations and scans the corresponding QR code, and automatically deleted after 14 days.
`clusterList`	idem	Within the backend server, this list contains all the `LTId` and timing information corresponding to a potential cluster. This list is public, it is downloaded by all the user terminals, and is updated each time a new cluster is identified. The cluster qualification happens when the hourly counter of a location exceeds a given threshold that depends on the location features.

Short name	Full Name	Description
`dupScanThreshold` (in seconds)	idem	Time tolerance in the duplicated scan mechanism: for a given `LTId`, a single QR code can be recorded in the localList every `dupScanThreshold` seconds. A similar check is performed on the server frontend.
`locationPhone`	idem	Phone number of the location contact person, stored as a set of 4-bit sub-fields that each contain a digit. This piece of information is only accessible to the manual contact tracing authority. It is meant to create a link between the digital system and the hand-written attendance register.
`locationRegion`	idem	Coarse grain geographical information for the location, in order to facilitate the work of the Manual Contact Tracing team. In case of France, it can contain a department number.
`locationPIN`	idem	Secret 6 digit PIN, known only by the location contact person, stored as a set of 4-bit sub-fields that each contain a digit. This piece of information is only accessible to the manual contact tracing authority. It is meant to create a link between the digital system and the hand-written attendance register.

3.2- Initial configuration of the service

Case of a location using a dedicated device(s) or tablet(s) and dynamic QR codes for synchronous scans The device(s) of a location must be initialized. This is either managed by the manufacturer in case of specialized device(s), before being used by the location manager, or by the location manager in case of tablet(s).

- The location keeps a long-term secret, `SK_L`, specific to this location, that is never communicated. This key is 408 bits (51 bytes) long, so that after concatenation with the `t_periodStart` and addition of 72 bits of padding, the total is 512 bits long and fits in a single SHA256 block size. If this location uses several devices, each of them must be configured with the same `SK_L`. With a dedicated device, this configuration can be done by the device manufacturer, meaning that the manufacturer is in charge of keeping this long-term secret. With a tablet, this is performed by the CLEA software used on the tablet, and when several tablets are used, a synchronization is required to make sure they all use the same long-term secret. Details are out of the scope of this document.
- Each device knows the public key of the Authority in charge of the backend server, `PK_SA`. When the deployment involves the MCT (options 1 and 2), the device also knows the public key of the Authority in charge of manual contact tracing, `PK_MCTA`.
- If the location has several totally independent rooms (e.g., a restaurant across two different buildings), distinct devices initialized with different long-term secrets, `SK_R1` and `SK_R2`, may be used in order to generate different location keys and identifiers.
- A pre-determined round hour (e.g., 4:00am, local timezone) is defined, that ideally corresponds to a moment when this location is closed. At that time, every day, the location `LTKey` and `LTId` are automatically renewed. Later, if the authority identifies a cluster in this location, it will be notified through this location temporary UUID, known only by the clients of the location who scanned the associated QR code.
- The case of a location that never closes should be handled in the most appropriate manner, for instance with a location `LTKey` and `LTId` renewal that corresponds to a low affluence period. The motivation is to reduce the probability that a client who arrives after the renewal and learns the new `LTId_{i+1}` via the new QR code, misses a cluster warning for the previous `LTId_i` since it started before the renewal.
- The case of a long event (e.g., over several days) requires specific attention. For instance, it can be a private event in a closed location during several days (e.g., a marriage or party over a week-end), with people who may only arrive on the second day, or leave earlier, or stay during the whole event. Such events are incompatible with a daily renewal of the location `LTKey` and `LTId`, since requiring all users to scan the new QR code after

each renewal is impractical. Having the location `LTKey` and `LTId` lasting the full duration of the event can be a practical solution. Then, since the exact moment when a participant definitively leaves the event is unknown by default, it can be preferable to use a coarse grain warning during the risk evaluation process.

- A period can also be shorter than 24 hours, when several activity periods are clearly defined, each of them separated by official closures of the location (e.g., the two services of a restaurant, at noon and in the evening). It is then possible to configure all the devices of the location with two or more renewal round hours (e.g., by the manufacturer). Those shorter periods are managed exactly in the same manner without any impact on the present specification. The main benefit of shorter periods is to avoid that a client of the noon service of a restaurant learns the potential cluster status of this location for the evening service, since this client will not know the new `LTId` used in the evening.
- The `periodDuration` parameter contains the chosen period duration, expressed in number of hours, between 1 and 255 (note that value 0 is invalid), the value 255 being reserved to the special case of a unlimited period duration. It is part of the information carried in the QR code and it is used by the server. This `periodDuration` determines the renewal frequency of the location `LTKey` and `LTId`. It is therefore a key parameter that defines the robustness against attackers who want to monitor the potential cluster status of a set of locations in the long term.
- An appropriate value for the `qrCodeRenewalInterval` parameter (duration after which a QR code is renewed) is chosen, depending on the device specifications and the desired protection against relay attacks. A value of 0 indicates the QR code is never renewed during the period, otherwise `qrCodeRenewalInterval` must be equal to a power of two. A default value is: `2^^10 = 1024 seconds` (approx. 17 minutes).

Case of a location manager or private event organizer who relies on a web service to generate a static QR code for synchronous scans It is also possible to use a web service to generate a static QR code. Here the whole generation process is done within the web browser, thanks to a dedicated javascript library: an `SK_L` secret key is generated locally, the `PK_SA` and potentially `PK_MCTA` public keys are also communicated to the web browser. No state is kept on the web service. The produced QR code is necessarily static, meaning that `periodDuration = 255` and `qrCodeRenewalInterval = 0`. With large locations, the manager can easily provide several prints of the same QR code. In any case, it is recommended to regularly generate and propose a new QR code in order to (slightly) reduce the attack probability and improve the user privacy.

3.3- Location Temporary Key (LTKey) and Location Temporary UUID (LTId) generation

Step 1: key generation A key is generated for the location. With a dynamic QR code, this is a temporary key which is automatically renewed (by default once a day) at a predefined round hour (e.g., at 4:00 am) which ideally corresponds to a closing time of the location. For the particular case of a static QR code, this key is in fact never renewed, the location manager needs to go to the website to renew the whole QR code, `SK_L` included. For the given period, this key is computed as follows:

```
LTKey(t_periodStart) = SHA256(SK_L | t_periodStart)
```

where `t_periodStart` is the reference timestamp for the beginning of the period, in NTP format (number of seconds since January 1, 1900). The `t_periodStart` value must match the predefined round hour: it cannot just be the result of a `gettimeofday()` (or similar) converted to an NTP time, a rounding to the nearest predefined round hour is necessary. For instance, 3h59mn48s and 4h00mn31s are both rounded to the same 4h00mn00 `t_periodStart` value, that is also necessarily multiple of 3600 seconds.

Step 2: UUID generation In order to keep this key secret with respect to the user, the device derives the following UUID from it:

```
LTId(t_periodStart) = HMAC-SHA-256-128(LTKey(t_periodStart), "1")
```

where HMAC-SHA-256-128 denotes the Keyed-Hash Message Authentication Code (HMAC) in conjunction with the SHA-256 cryptographic hash function and whose output is truncated to 128 bits, as defined in **RFC4868** and **RFC2104**.

Indeed, an attacker who scans a QR code must not know the `LTKey(t_periodStart)` key in order to prevent him from being able to forge a new QR code in place of the device.

If there are several devices, each of them must generate the same `LTKey(t_periodStart)` and `LTId(t_periodStart)` when switching to a new period. This is guaranteed if the same `t_periodStart` value is generated. The renewal at a pre-defined well-known full hour, plus a limited drift of the devices (e.g., one or two minutes per year) guarantees this. The fact the devices are not perfectly synchronized (because of different clock drifts across devices), a small hazard is possible (i.e., some devices will still display the old QR code and others the new one). It does not have any consequence if the location is closed to public at that moment.

3.4- QR code content

High level view of the QR code content The QR code of a location contains a URL ("deep link") structured as follows:

```
"country-specific-prefix" "Base64url(location-specific-part)"
```

The country specific prefix is: `https://tac.gouv.fr?v=0#` in case of France, where `v=0` indicates it's the protocol version 0 of CLEA, and the `#` character prevents the text that follows (namely the Base64url encoding of the location specific part) to be sent to the `tac.gouv.fr` server if the application is not already installed on the user terminal.

The remaining of this section defines the location specific part.

With a dynamic QR code, this QR code is renewed when switching from one period to another (change of `LTKey` and `LTId`), but also periodically during the period. This renewal during the period happens every `qrCodeRenewalInterval` seconds. It is a balance between the calculation and autonomy constraints of the device on the one hand (the higher the `qrCodeRenewalInterval` value, the better), and the desired protection against relay attacks on the other hand (the lower, the better). Note that if there are several devices, an asynchronism between them during the renewal of the QR code does not pose any problem: if the `t_qrStart` values may differ, the `LTId(t_periodStart)` will remain identical. With a static QR code, the `LTKey` and `LTId` are kept unchanged for an undefined duration.

With the current specification for CLEA protocol version 0, a single `location-specific-part` (LSP) type is defined:

- `LSPtype = 0`: for a QR code compatible with a synchronous scan (i.e., when entering a location). The QR code may either be static or dynamic, and in the latter case associated to a freshness check. The check-in timestamp is the time when the user scans this QR code, called `t_qrScan` hereafter.

More precisely, it is structured as follows (high-level view):

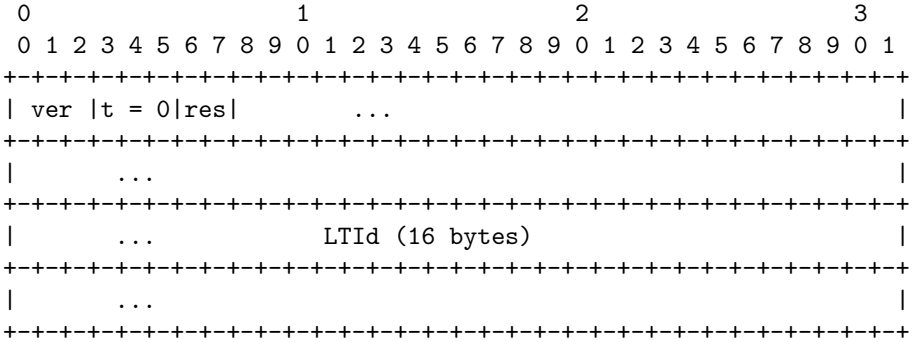
```
LSP(t_periodStart, t_qrStart) = [ version | LSPtype = 0 | reserved1 | LTId(t_periodStart)
    | Enc(PK_SA, msg) ]
```

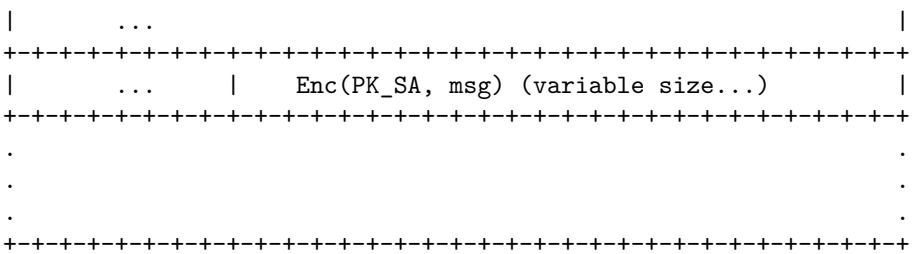
where:

```
msg = [ staff | locContactMsgPresent | reserved2 | CRIexp | venueType | venueCategory1 |
    | venueCategory2 | periodDuration | ct_periodStart | t_qrStart | LTKey(t_periodStart)
    | Enc(PK_MCTA, locContactMsg) if locContactMsgPresent==1 ]
```

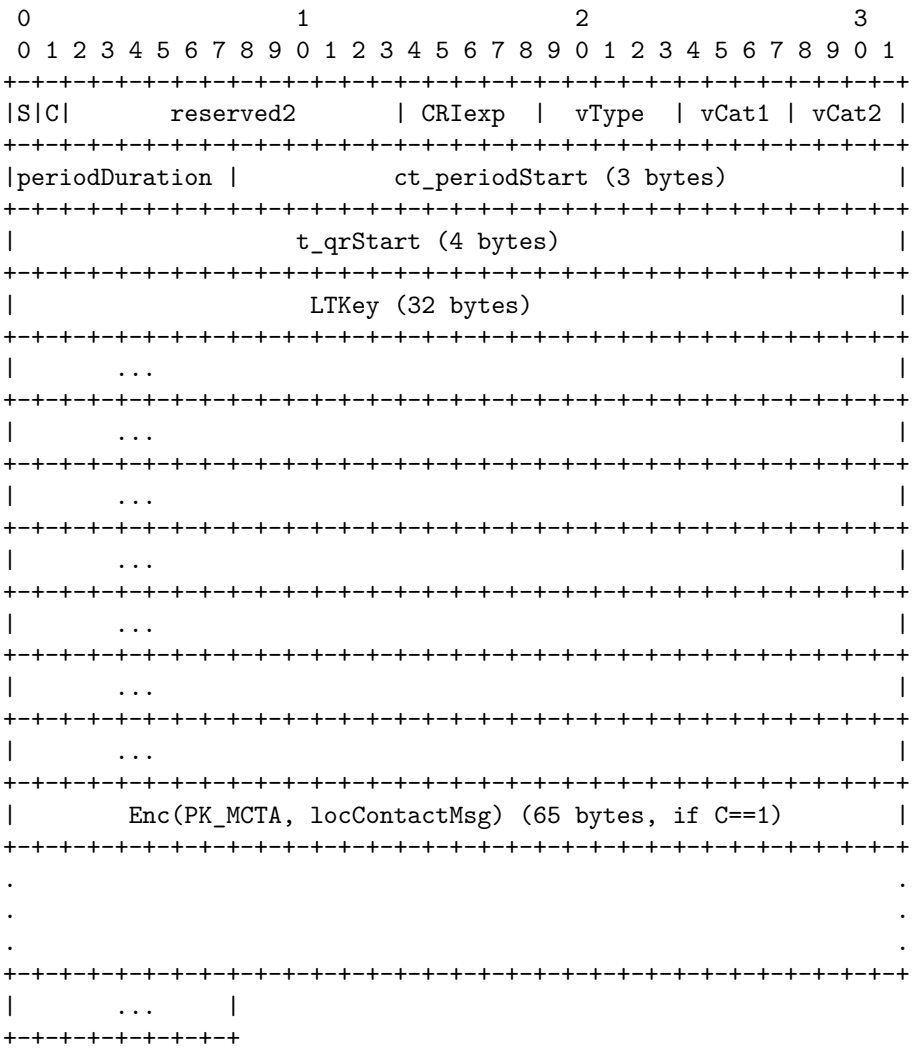
The various fields are described below. The `Enc(PK_MCTA, locContactMsg)` is defined in section "A user tested COVID+ has used the CLEA system".

Binary format of the location-specific-part for LSPtype = 0 (synchronous scan) The following binary format must be used when `LSPtype = 0`:





Binary format of the msg The following binary format for the `msg` message must be used:



Field descriptions The "big endianness" (also called "network endianness" with the Internet protocol suite), transmitting the most significant byte first, must be used whenever meaningful. The location specific part contains (in plaintext or encrypted) the following fields, in this order:

- `version` (3 bits) (`ver` in figure): this is the protocol version number, in order to enable an evolution of the protocol. The present specification corresponds to protocol version 0.
- `LSPtype` (3 bits) (`t=0` in figure): this is the LSP type 0 used with a QR code compatible with a synchronous scan (i.e., when entering a location).
- `reserved1` (2 bits) (`res` in figure): this field is unused in the current specification and must be set to zero.
- `LTId` (16 bytes, or 128 bits): this field carries the location temporary UUID for the period.

- `staff` (1 bit) (`S` in figure): this field, when equal to 0, indicates a regular QR code, for regular users, and when equal to 1, indicates a QR code specific to a staff member of the location.
- `locContactMsgPresent` (1 bit) (`C`in figure): this field, when equal to 1, indicates the `locContactMsg` is used and present in the QR code, and when equal to 0, indicates it is absent. It follows that the `msg` size can largely vary, depending on the use or not of a `locContactMsg`.
- `reserved2` (12 bits): this field is unused in the current specification and must be set to zero.
- `CRIexp` (5 bits): this field enables to communicate the `qrCodeRenewalInterval` value in a compact manner, as the exponent of a power of two. If this field contains the value `0x1F` (maximum value for a 5 bit field), `qrCodeRenewalInterval` must be set to `0` in order to indicate the QR code will not be renewed during the whole period (no QR code renewal). Otherwise, `qrCodeRenewalInterval` must be set to the value `2^^CRIexp` seconds. It follows that:

```
qrCodeRenewalInterval = (CRIexp == 0x1F) ? 0 : 2^^CRIexp; // value in seconds
```

It means the QR code renewal will happen after an interval that is comprised between `1` and `2^^30` seconds inclusive, or never (if `qrCodeRenewalInterval == 0`). Of course, a new QR code must be generated at the start of a new period (because the `LTKey` and `LTId` fields change) even if the `qrCodeRenewalInterval` is not finished.

- `venueType` (5 bits) (`vType` in figure): this field specifies the type of the location/venue (e.g., a restaurant). The encoding is country specific (for instance, for France, it can be mapped to the Types d'ERP classification).
- `venueCategory1` (4 bits) (`vCat1` in figure): this field specifies a first level of venue category. This is an opaque field whose semantic is out of scope of the present document.
- `venueCategory2` (4 bits) (`vCat2` in figure): this field specifies a second level of venue category. This is an opaque field whose semantic is out of scope of the present document.
- `periodDuration` (1 byte): this field contains the duration, in terms of number of hours, of the period. Since this period duration is location dependent, this information needs to be communicated to the server. The value 255 is reserved to the special case of an unlimited period duration. The default value is 24 hours (a period per day), but different values may be defined up to a maximum of 254 hours (i.e., 10 days and 14 hours).
- `ct_periodStart` (24 bits): this field contains the starting time of the period in a compressed manner, dividing `t_periodStart` by 3600, which is guaranteed to be an integral value since a period necessarily starts at a round hour (i.e., is multiple of 3600). It follows that:

```
t_periodStart = ct_periodStart * 3600;
```

- `t_qrStart` (32 bits): this field contains the starting time of the QR code validity timespan, expressed as the number of seconds since January 1st, 1900 (NTP timestamp limited to the 32-bit seconds field).
- `LTKey` (32 bytes, or 256 bits): this field carries the location temporary key for the period.

Encryption and integrity protection The `msg` message must be encrypted using the ECIES-KEM [**ISO18033-2**] [**Shoup2006**] [**Libecc**] hybrid encryption scheme that provides both confidentiality, using an asymmetric encryption scheme, and integrity verification. More precisely, this scheme uses SECP256R1 ECDH as KEM, KDF1 using SHA256 hash as KDF and AES-256-GCM with a fixed 96-bits IV as DEM and TAG. To the original `msg` message, the hybrid ECIES-KEM scheme appends a block of 49 bytes that contains both a tag and an ephemeral public key. A detailed description is given in Appendix A.

While only the `msg` message is encrypted, the integrity protection encompasses the whole `LSP` message, including the cleartext part of the LSP: any accidental or malicious modification is therefore automatically detected.

Size of the various QR codes A Level 12 65x65 QR code Type 2 (see Section 2.4) has a limited capacity, 287 binary characters for redundancy level M or 203 for redundancy level Q. It is therefore essential that the full "deep link" complies with these limits. The country specific prefix, namely `https://tac.gouv.fr?v=0#` in case of France, requires 24 characters.

The location specific part size depends on:

- the optional presence of the `locContactMsg` message (when the `locContactMsgPresent == 1`), which consists of 16 bytes of cleartext.

It should also be noted that :

- the hybrid ECIES-KEM encryption add a 49-byte overhead;
- the Base64url encoding increases the size by a factor `1.33` approximately.

The following table summarizes the situation.

Name	size with LSP Type 0
`https://tac.gouv.fr?v=0#` prefix (characters)	24 chars
Plain text part of the LSP (bytes)	17 bytes
msg part of the LSP, without `locContactMsg` (bytes)	93 bytes
`locContactMsg` size (bytes)	65 bytes
size with `locContactMsg` before Base64url encoding (bytes)	175 bytes
size with `locContactMsg` after Base64url encoding (chars)	234 chars
total URL size with `locContactMsg` (characters)	***258 chars***
size w/o `locContactMsg` before Base64url encoding (bytes)	110 bytes
size w/o `locContactMsg` after Base64url encoding (chars)	147 chars
total URL size w/o `locContactMsg` (chars)	***171 chars***

3.5- Synchronous scan of a QR code when a client enters a location (LSP Type 0)

A client entering a location scans the QR code, and the CLEA application adds the following tuple to its local list, `localList`, which records the visited locations:

```
{QR_code, t_qrScan}
```

where `t_qrScan` is the timestamp in NTP format (32-bit seconds field) of the CLEA application. Entries in the local list are automatically removed after 14 days.

Detection of duplicated scans by the CLEA application Before adding `{QR_code, t_qrScan}` in the local list, the CLEA application checks that an entry with the same `LTId` is not already there, with a scanning time "close" to `t_qrScan`:

```
// assume a previous entry already exists for the same LTId, with a scanning time t_scan0
if (abs(t_qrScan - t_scan0) > dupScanThreshold) {
    // record the new entry, scanned sufficiently later after the previous scan for this LTId
} else {
    // reject the new entry as duplicated
}
```

where the `dupScanThreshold` is the time tolerance in the duplicated scan mechanism: for a given `LTId`, a single QR code can be recorded in the localList every `dupScanThreshold` seconds.

This verification is intended to avoid disrupting the "cluster" qualification mechanism by artificially increasing the number of reports for a given time slot and location, which may be accidental (a client unwillingly scans twice the QR code) or deliberate (a malicious client, who knows he will likely be tested COVID+). From this point of view a large value for `dupScanThreshold` is preferable. However, a regular client of a restaurant that does not distinguish between the noon and evening services (e.g., it remains continuously open from 11:00am and 12:00pm) will need to scan and register in its `localList` two QR codes with the same LTId on that day, one for lunch and another one for dinner. Choosing an appropriate value of closeness for this check is key. By default, a value of 3 hours is used:

```
dupScanThreshold = 3 * 3600;
```

Note that this is not an absolute protection as an attacker using a malicious application could easily bypass this check. Note also that having a `dupScanThreshold` value that depends on the location specificities (e.g., the expected duration during which a client is supposed to stay in a location) is not feasible since this piece of information is in the encrypted part of the QR code and is not accessible to the CLEA application.

Reliability of the t_qrScan timestamp The replay protection is limited by the availability of a trustworthy `t_qrScan` timestamp, which garanties that the local terminal clock has not been maliciously modified to match that of the replayed QR code.

Although nothing can prevent a malicious application from storing a specially crafted timestamp, the official application should propose a trustworthy internal clock to be used for this purpose. The accuracy of this trustworthy clock needs to be in line with the `qrCodeRenewalInterval`. With an interval of `2^^10 = 1024 seconds`, the accuracy requirement is pretty low. The CLEA application benefits from such an internal trustworthy clock, making it relatively robust in front of such a relay attack.

3.6- Upload of the location history by a client tested COVID+ and cluster detection on the server

Let us assume the user has been tested COVID+. In that case, her CLEA application asks for her explicit informed consent to upload her location history. If the user explicitly agrees, the following operations take place.

Processing of the user location history by the frontend server The user application uploads to the server, within a TLS connection, the location history stored in its local list, `localList`, along with the associated authorization, meant to prove the user has indeed been tested COVID+. The details of this authorization mechanism are out of scope of the present document.

The location history consists of a set of records of the form:

```
{QR_code_0, t_checkin_0}, {QR_code_1, t_checkin_1}, {QR_code_2, t_checkin_2}...
```

where the `t_checkin` is the scanning timestamp (LSP Type 0).

This history is by design limited to 14 days of history. It could be further restricted, or the uploaded data could add additional information. For instance, if the goal is to do forward tracing, and if the user experienced symptoms starting from a known date, it could be helpful to take advantage of the start of the "infectious period" (i.e., when the user could contaminate others) to remove records prior to that date. On the opposite, if the goal is to do backward tracing, it could be helpful to distinguish between the "infectious period" (when the user could contaminate others) and "infected period" (when the user has potentially been contaminated), when known. The details of what to do exactly, as they depend on the Health Authority decisions, are out of scope of this specification.

The frontend of the server:

- first of all verifies the COVID+ status of the user and discards an invalid upload from a user who does not show a valid authorization.
- then it checks that this history does not contain duplicated scans, using the same methodology as before, namely by checking if: `(abs(t_qrScan - t_qrScan0) > dupScanThreshold)`. If any duplicated scan is identified (test is true), it is recommended to discard the whole history as coming from a invalid application. This verification is meant to protect the server against malicious applications that could try to bypass the local duplicated scan check.
- the frontend then sanitizes the message (e.g., by removing the source IP address).
- finally the frontend mixes each entry from this user with that of other other COVID+ uploads in order to minimize privacy risks.

Processing of a user location record by the backend server When receiving a given `{QR_code, t_checkin}` tuple (they are processed independently from one another as a result of the frontend mixnet), the backend server:

- Step 1: decrypts the `msg` part of the QR code, using its `SK_SA` secret key, and checks the message integrity. In case of problem, the server rejects the tuple.

- Step 2: With an LSP Type 0 QR code (synchronous scan), if `qrCodeRenewalInterval > 0`, a freshness check is performed for this tuple in order to limit relay attacks. In that case, the `t_checkin` of this tuple is the QR code scan timestamp, `t_qrScan`. So, if `t_qrScan` (generated by the CLEA application during the scan) and `t_qrStart` (generated by the device and protected from malicious modifications by being in the encrypted part of the QR code) are "too different", the server rejects the tuple. The tolerance depends on the `qrCodeRenewalInterval` value, on the possible drift of the device clock (e.g., one or two minutes per year), and on the accuracy of the CLEA application clock on the user terminal. For instance it checks that:

```
| t_qrScan - t_qrStart | < qrCodeRenewalInterval + 300 sec + 300 sec
```

in order to take into account the possibility of scanning the code just before its renewal, including a maximum drift of 5mn for this device compared to the official time, and also a maximum drift of 5mn for the CLEA clock.

This verification is intended to limit (without being able to totally prevent them) relay attacks where the attacker scans a QR code from a target location and communicates it to several supposed patients in order to create a fake cluster afterwards. The attack is thus limited in time to the defined tolerance.

If `qrCodeRenewalInterval == 0`, there is no freshness check, the QR code being static during the whole period (e.g., a day).

- ***Step 3:*** computes from the information stored in the encrypted `msg`, `HMAC-SHA-256-128(LTKey(t_periodStart), "1")`, and compares its value to the `LTId` value retrieved from the unencrypted part of the QR code. If the two values differ, the server rejects the tuple.

- ***Step 4:*** depending on the location category, the server determines the corresponding exposure time (e.g., 3 hours of presence in the case of a restaurant to take into account the fact that periods of presence are rounded up to the hour). In the case of a QR code of type "staff" (indicated by the flag `staff` set to 1), the exposure time is to be considered from `t_qrScan` and until the end of the period.

- ***Step 5:*** The server stores the exposure as follows. If `LTId` has already been flagged as exposed, it retrieves the associated context, otherwise it creates a new context for `LTId`, for example in a record of the following type:

```
typedef struct {
        LTId_t          LTId;                   // for that LTId (a hashtable can be used)
        uint32_t        t_periodStart;          // start time (in seconds) of the period
        uint8_t         periodDuration;         // duration in terms of hours, also the number
                                                // of entries in the hourlyExposureCount table
                        // if lower than 255.
        uint8_t         hourlyExposureCount[];  // number of COVID+ users per hour
} LTId_exposure_context;
```

where the first three fields are initialized thanks to the corresponding fields in the QR code (after conversion for `t_periodStart`). The `hourlyExposureCount[]` table should be large enough to encompass the whole duration of a very long event, when `periodDuration` is equal to 255 (possibly by using a different data structure, a list instead of table). The technical details are out of scope of the present document.

Then, if the exposure is 3 hours (previous example of the restaurant), the server calculates the index of the first hour of exposure:

```
h1 = floor((t_qrScan - t_periodStart) / 3600);
```

and increments the three hourly exposure counters of `LTId` (assuming `e` is a pointer to the appropriate context entry of type `LTId_exposure_context`):

```
for (uint32_t i = 0; i < 3; i++) {
        uint32_t        h  = h1 + i;
        if (h >= e->periodDuration)
                break;       // beyond the period end, stop immediately
        if (e->hourlyExposureCount[h] < 255)
                e->hourlyExposureCount[h]++;
}
```

It is important to verify that any index is within 0 and `e->periodDuration - 1` (inclusive) before updating any counter, since the 3 exposure hours (previous example) may extend beyond the closure of the location. The above code avoids wrapping to zero when a counter already reached its maximum value, 255 (counting above 255 is of course possible after changing the data type).

- ***Step 6:*** if the `e->hourlyExposureCount[h1]` (for instance) goes above the cluster qualification threshold (this threshold may depend on the location category and capacity), it adds `{LTId, h1}` to its `clusterList`, a public list periodically downloaded by all terminals. This list needs to be structured in batches, in order to make possible the partial download of a subset of it by terminals (see below). A threshold equal to 1 is likely to accelerate the cluster

identification process. Several levels of severity could also be considered depending on the exposure counter value. The details on how to exploit the various exposure counters are related to epidemiological policy considerations and are therefore out of scope of the present document.

3.7- Incremental downloads of the clusterList

The `clusterList` is made available by the server to all terminals, for instance via a Content Delivery Network, CDN, service. The `clusterList` information is structured in a manner that enables a terminal to download the entries in an incremental manner (rather than the 14 days content at once). Therefore, a terminal that fetches the list every day only downloads the latest entries, corresponding to new cluster locations identified since the previous fetch. This approach contributes to limit traffic overhead as well as the required amount of processing and storage on the terminal.

To that goal the following data structure is used (inspired from GAEN, see Kessibi2020):

- the `clusterList` is split into a collection of files, each of them corresponding to a given time span (e.g., 6 hours) and collecting all the new cluster locations identified by the server during this time span;
- a well-known URL is defined, for instance: `https://tacw.example.com/clusterlist/`, that is meant to contain the various files of the `clusterList` collection;
- the server makes available a well-known index, `index.txt`, that lists the various files available, in a chronological order, and limited to a 14 days history;
- each file contains some metadata and the identification of all the clusters added to the `clusterList` during that time span (LTId and timing information);
- the file names use the following convention: `cluster_file_ID_DATE.json` where `ID` is a monotonically incremented identifier, starting at 0 when bootstrapping the system, and the `DATE` suffix indicates the corresponding `yyyymmdd` (it is essentially here to facilitate human checks);
- all files are made available as soon as possible in order to quickly let users know if they are at risk. It follows that several files per day should be made available (4 in this example).

Here is an example of `index.txt` file (usually there are as many entries as required to cover the 14 days window):

```
cluster_file_521_20210215.json
cluster_file_522_20210215.json
cluster_file_523_20210215.json
cluster_file_524_20210215.json
cluster_file_525_20210216.json
cluster_file_526_20210216.json
```

Here is an example of `cluster_file_521_20210215.json` file (2 clusters only are listed, corresponding to time span 0am-6am UTC time):

```
{
    clusterListExport: {
        start: 3822336080,
        end: 3822357680,
        signature_infos: {
            TBD
        }
    },
    clusterInfo: [
        {
            TLId: "put-here-the-resulf-of-base64url-encoding-of-TLId",
            clusterStart: 3822346880,
            clusterDuration: 2
            warningLevel: 1
        },
        {
            TLId: "put-here-the-resulf-of-base64url-encoding-of-TLId",
            clusterStart: 3822354080,
            clusterDuration: 3
```

```
            warningLevel: 3
        }
    ]
}
```

where:

- `clusterStart` is the round hour from which the location is considered cluster, using NTP time.
- `clusterDuration` is the number of hours, starting at `clusterStart` (included), where the location is considered as a cluster.
- `warningLevel` determines the severity of the warning, from "low" (1), "medium" (2), to "high" (3). The exact criteria defining the severity of a warning are out of scope of the present document.

Note that Unix timestamps (that uses an epoch located at 1/1/1970-00:00h (UTC)) and NTP (that uses 1/1/1900-00:00h) timestamps can be converted to one another by adding or substracting a fixed number of seconds, corresponding to a fixed offset equivalent to 70 years in seconds (there are 17 leap years between the two dates)[4]:

```
(70 * 365 + 17) * 86400 = 2208988800
```

3.8- Decentralized risk analysis in the CLEA application

Each CLEA application periodically downloads the `clusterList` from the server, in an incremental manner. This is achieved by downloading the `index.txt` file first, identifying the newly available files (it is assumed the application remembers what is the latest file name downloaded). The CLEA application then downloads each of the new files, remembers the name of the last one, and processes them one by one.

Then the CLEA application checks locally if there is one or more intersections between:

- the information `{LTId, t_checkin}` from each tuple of its `localList` (the `LTId` is available in clear text in the QR code scanned in order to allow this comparison).
- the information `{LTId_cluster, h1_cluster}` from the downloaded `clusterList`. In case of a match, the application informs the user by means of a warning, indicating for instance the associate date. It the server provide a certain degree of risk (i.e., distinguishes low and high risks), this information is communicated to the user. However, since the `{LTId_cluster, h1_cluster}` information is public, a curious user may be able to know more about the exact time of exposure.

3.9- Linking the CLEA digital system and the hand-written attendance register

The use of the CLEA digital system is based on a voluntary decision of the user, the alternative consisting for this user in leaving her name in the hand-written attendance register. Consequently, a link between the two systems should be established. The following sections explain how this can be done, depending on whether a user tested COVID+ has used the CLEA system or the hand-written attendance register.

It should also be noted that there are use-cases where the hand-written attendance register may not exist, for instance in case of digital ticketing. In that case, the `locContactMsg` should be ignored, by setting the `locContacMsgPresent` flag to 0. Similarly, the Health Authority may decide not to link the two systems together, in which case the `locContacMsgPresent` flag should be set to 0.

It should also be noted that the link between the two systems is not perfect. If the cluster qualification threshold is strictly superior to `1`, it can happen that a given location should be qualified as cluster because the total number of COVID+ persons who were there at the same time is sufficient, but no alert is raised because some of them used the CLEA application, and the others the attendance register.

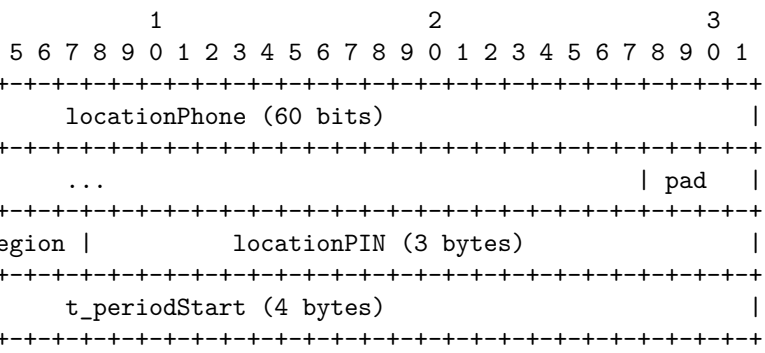
A user tested COVID+ has used the CLEA system In that case, the backend server qualifies as a cluster a given location, based on an uploaded QR code (and perhaps previous ones depending on the threshold). Since the re-identification of the location is the responsibility of the authority in charge of the manual contact tracing, assumed different from the authority in charge of the backend server, the backend server communicates through a TLS connection the location contact re-identification part of the QR code, encrypted via the public key of the Manual Contact Tracing Authority, along with cluster timing information.

[4] See: [https://stackoverflow.com/questions/29112071/how-to-convert-ntp-time-to-unix-epoch-time-in-c-language-linux]

The `locContactMsg` message is structured as follows (high-level view):

```
locContactMsg = [ locationPhone | padding | locationRegion | locationPIN | t_periodStart ]
```

The following binary format must be used:



- `locationPhone` (60 bits): this field contains a phone number, where each digit is stored one by one in a 4-bit nibble. The phone number must be encoded using the E.164 standard that requires phone numbers to have a maximum length of 15 digits. For instance, in case of France, `+33 1 02 03 04 05` will be stored as (binary) `0011 0011 0001 0000 0010 0000 0011 0000 0100 0000 0101 1111 1111 1111 1111`. Unused nibbles must contain the `1111` / `0xF` value.
- `padding`(4 bits) (`pad` in figure): this field is unused in the current specification and must be set to zero.
- `locationRegion` (1 byte): this field contains coarse grain geographical information for the location, in order to facilitate the work of the Manual Contact Tracing team (e.g., for countries that rely on a regional organisation, it enables the cluster record to be routed directly to the right regional Manual Contact Tracing team). In case of France, it can contain a department number.
- `locationPIN` (3 bytes): this field contains a 6-digit secret PIN known only by the location contact, communicated when registering to the device manufacturer or on the web site when generating a static QR code. It is meant to prevent an attacker who knows the contact phone number of a target location (this phone number is usually public) to forge a new QR code and handle it to a user tested COVID+. Thanks to the `locationPIN`, the manual contact tracing team can check the QR code validity with the location contact: if the two pin codes do not match, the QR code is reputed invalid and ignored (note that the CLEA users have no risk, the forged `LTKey` and `LTId` being totally distinct from the ones actually used in this location).
- `t_periodStart` (4 bytes): Starting time of the period. With a dynamic QR code, including the `t_periodStart` field in the `locContactMsg` has a major benefit: it prevents the `Enc(PK_MCTA, locContactMsg)` to remain constant over the time for this location. Therefore, the authority in charge of the backend server cannot re-identify this locations across different periods, the encrypted message changing altogether as soon as the `t_periodStart` changes. If the location chooses to use a static QR code, this protection is of course meaningless, the location pseudonym remaining constant by definition.

As soon as the authority in charge of the manual contact tracing receives the encrypted message, it decrypts it and checks its integrity. Then the authority informs the location contact person, checking the PIN code first, and asking this latter to communicate the content of the hand-written attendance register for the appropriate period. The details of how this is done is out of scope of the present document.

A user tested COVID+ has used the hand-written attendance register Here, the Manual Contact Tracing team determined that a certain location, at a certain time, should be qualified as cluster. However, since the person(s) tested COVID+ used the hand-written attendance register of the location, there is no scanned QR code that could be used to trigger the cluster qualification at the backend server. In order to make it possible, the Manual Contact Tracing team needs to ask the location manager to recover and upload the QR code of this period. For instance, the team can physically visit the location, discuss with the manager and help her upload her own scanned QR code of that day, using a dedicated authorization token. In case of a static QR code, obtaining the required QR code is not an issue. In case of a dynamic QR code, this scenario requires that the location manager scans her location QR code everyday, as any employee is supposed to do. Although there is a risk that she omitted to do so on that day (thereby preventing a notification through the CLEA applications), the probability this happened is reasonable.

3.10- Management of the location employees

The employees of a location should be able to benefit from the service in order to be warned if their workplace is a cluster, or to upload to the server that an employee has been tested COVID+. Since they have a long presence in the location, the employees must scan a specific QR code which differs from regular QR codes by the "staff" flag set to 1. Such an "staff" QR code can be generated through an appropriate manipulation on the box (e.g., using a magnetic badge).

1. to warn an employee that their workplace is a cluster: As the risk analysis is local, it is sufficient to use a dedicated "extended presence range" mode in their CLEA application. The verification is then done based on this extended presence range starting from the scan time. However, an employee who has two disjoint working periods may be warned whereas the cluster corresponds to a non working period. To address this issue, the employee can obtain more precise information from the application on the times when the location was declared as a cluster in order to perform a manual verification (the detailed information is anyway public).
2. to report that an employee has been tested COVID+: An employee tested COVID+ must upload their history to the server as a regular client would: `{QR_code_0, t_scan_0}, {QR_code_1, t_scan_1}, {QR_code_2, t_scan_2}`. The fact that each QR code has the "staff" flag set to 1 tells the server that the exposure must be considered over the entire remaining time range from `t_scan_0`. This conservative operation ignores the case of an employee with a complex schedule (disjoint working periods) which would require a more precise mechanism with badging when entering and leaving the location.

Limit: a "staff" QR code is incompatible when `periodDuration == 255` (unlimited period duration). This is a consequence of the notion of "extended presence range" that cannot be used for the whole period if this latter is unlimited.

Limit: a "staff" QR code creates a risk of attack amplification since the attacker could switch a place as a cluster during the whole period and not the typical time slot of that place (e.g., 3 hours for a restaurant). It is recommended to allow this feature only on a device located in a safe place of the location in order to avoid that a simple client can unblock it.

4- Conclusions

The CLEA cluster exposure verification protocol features key privacy properties thanks to its local risk evaluation (the list of scanned QR codes remains on the user's smartphone) and its totally anonymous approach (no user pseudonym, no user-related information kept in the backend server). Only users tested COVID+ upload information to the central server (their list of scanned QR codes with a scanning timestamp, without any user identifying information), on a voluntary basis: this upload is required in order to identify potential clusters and inform other users.

Flexibility is another key design requirement. Indeed, the CLEA protocol can be used in a dynamic manner, QR code being renewed periodically which leads to location temporary pseudonyms (`LTId`). Having temporary pseudonyms is key to avoid most practical attacks attempting to collect these pseudonyms on a large scale (for instance to display geolocated maps of cluster locations). However this dynamic approach requires the location to have one or more devices (depending on the location size) in order to display such dynamic QR codes. Not all locations are concerned by such attacks. Therefore a location manager or a private event organizer may prefer the practical and device-free static variant of CLEA, where a QR code has a permanent validity: a QR code will be generated using a dedicated web service, and then be printed and displayed in the location.

Another key design requirement is the ability to use embedded, autonomous devices, that are:

- battery powered (an autonomy of several months is expected);
- feature an limited screen size and resolution (e.g., a maximum of 196 printable characters can be encoded into the QR code);
- have no network connection, neither wired nor wireless.

This opens the road to cheap, fully dedicated, and secure devices, meant to facilitate the practical deployment of the CLEA system in commercial locations (once installed, the device can be forgotten).

The CLEA system being totally anonymous, mitigating replay attacks is needed. This attack consists, for the attacker, to scan a target location QR code and relay it to a potentially COVID+ user in the hope to trigger a cluster decision. If a static CLEA deployment is vulnerable, the dynamic version includes counter-measures, limiting

the QR code validity duration and including automatic timing information when a QR code is scanned, which also requires the application to provide a trustworthy clock. Although not perfect (a modified CLEA application could easily bypass the protection, and the QR code validity period is anyway limited by the ability for an embedded device to renew it frequently), it is considered a reasonable protection.

Finally, the CLEA system enables the authority in charge of the backend server to gather some basic statistics on the pandemic and the potential efficacy of the system: number of locations that triggered a warning on a daily basis and typology of these locations (see the `venueType`, `venueCategory*` fields). However, the risk being assessed locally, by default, the authority will not know the number of people warned.

5- References

[**Lueks20**] W Lueks, S. Gürses, M. Veale, E. Bugnion, M. Salathé, C. Troncoso, "CrowdNotifier: Decentralized privacy-preserving presence tracing", version 8, Octobre 2020.

[**Kessibi2020**] G. Kessibi, M. Cunche, A. Boutet, C. Castelluccia, C. Lauradoux, V. Roca, "Analysis of Diagnosis Key distribution mechanism in contact tracing applications based on Google-Apple Exposure Notification (GAEN) framework", Sept. 2020. [https://hal.inria.fr/hal-02899412v4]

[**RFC2104**] H. Krawczyk, M. Bellare, R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", Request for Comments (RFC) 2104, Standards Track, February, 1997. [https://tools.ietf.org/html/rfc2104]

[**RFC4868**] S. Kelly, S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", Request for Comments (RFC) 4868, Standards Track, May, 2007. [https://tools.ietf.org/html/rfc4868]

[**ISO18033-2**] "ISO/IEC 18033-2:2006: Information technology — Security techniques — Encryption algorithms — Part 2: Asymmetric ciphers", May 2006. [https://www.iso.org/standard/37971.html]

[**Shoup2006**] V. Shoup, "ISO 18033-2: A Standard for Public-Key Encryption", 2006. [https://www.shoup.net/iso/]

[**Libecc**] R. Benadjila, A. Ebalard, J-P. Flori, et al., "Libecc project", Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI), France, 2017. [https://github.com/ANSSI-FR/libecc]

[**QRcodeWeb**] "What is a QR code (QRcode.com official web site)", [https://www.qrcode.com/en/about/version.html]

[**QRcode18004**] "Information technology — Automatic identification and data capture techniques — Bar code symbology — QR Code", ISO/IEC 18004, First edition 2000-06-15, 2000.

[**RFC4648**] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings", Request for Comments (RFC) 4648, Standard Track, October 2006. [https://tools.ietf.org/html/rfc4648]

A- Description of the hybrid encryption scheme and the `Enc` and `Dec` functions

A.1- Notations

Symbol	Description
`p`	Prime number, dimension of `F_p` the finite field over which SECP256R1 is defined
`G`	Base point of SECP256R1
`n`	Order of `G`
`S`	Shared secret
`K`	Derived key for symmetric encryption
`IV`	AES-GCM IV set to the 96-bit constant value `0xF01F2F3F4F5F6F7F8F9FAFB` (big endian encoding)
`C0`	Ephemeral public key

A.2- Pseudo-code:

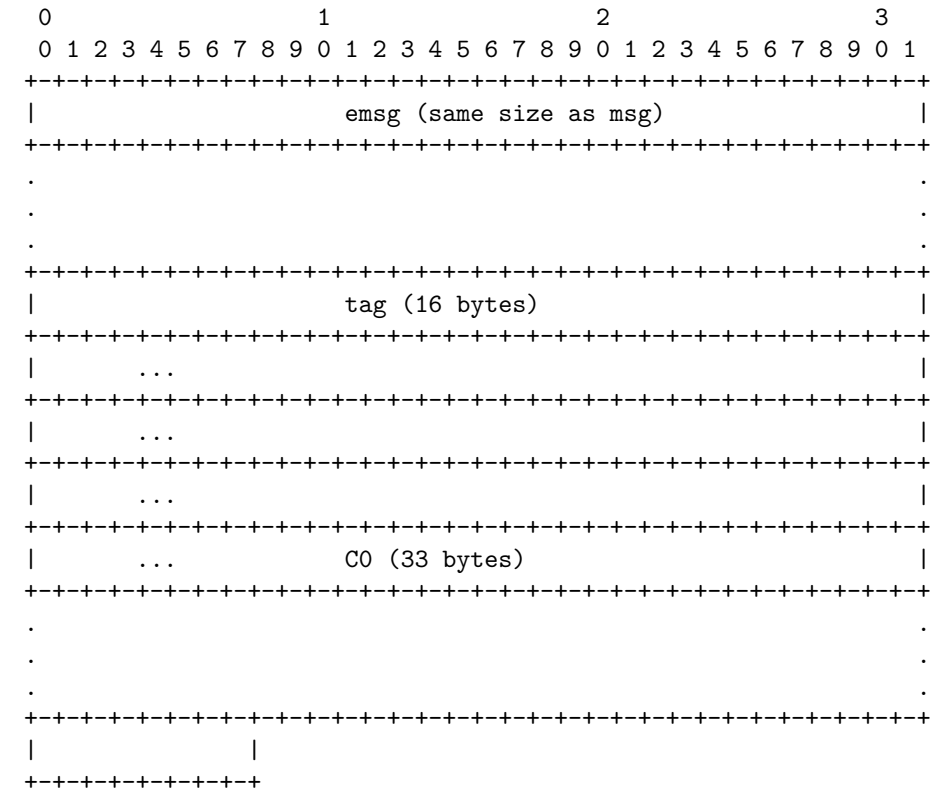
```
Enc(pub_key, msg):
    -Draw an ephemeral private key r in [1, n-1]
    -Compute C0 = r * G
```

```
    -Compute S = r * pub_key
    -Compute K = KDF1(C0 | S)
    -Compute emsg = AES-256-GCM(K, IV, msg) and tag = GMAC(K, IV, emsg)
    -Return (emsg, tag, C0)

Dec(priv_key, emsg, tag, C0):
    -Compute S = priv_key * C0
    -Compute K = KDF1(C0 | S)
    -Compute msg = AES-256-GCM(K, IV, emsg) and tag' = GMAC(K, IV, emsg)
    -if(tag == tag') return msg else raise error
```

Note that in computation of K with the KDF1 function C0 is represented in its compressed form as specified in ANSI X9.62 (i.e. 33 bytes) and S is represented by its X coordinate (i.e. 32 bytes)

The encrypted message out of the `Enc` function must have this format:



The size of the plain text message is therefore increased by 49 bytes, corresponding to the tag and ephemeral public key.

Note that the ephemeral public key C0 is a point of SECP256R1 curve, it is stored in its compressed form as specified in SEC1